SIM2SPICE, A TOOL FOR COMPILING SIMULINK DESIGNS ON FPAA AND APPLICATIONS TO NEUROMORPHIC CIRCUITS

A Thesis
Presented to
The Academic Faculty

By

Csaba Petre

In Partial Fulfillment
of the Requirements for the Degree
Master of Science in Electrical Engineering

School of Electrical and Computer Engineering
Georgia Institute of Technology
December 2009

SIM2SPICE, A TOOL FOR COMPILING SIMULINK DESIGNS ON FPAA AND APPLICATIONS TO NEUROMORPHIC CIRCUITS

Approved by:

Dr. Paul E. Hasler, Advisor
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. David V. Anderson
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Christopher J. Rozell
School of Electrical and Computer Engineering
Georgia Institute of Technology

Date Approved: November 2009

ACKNOWLEDGMENTS

I would like to thank my advisor, Professor Paul Hasler for his guidance, help, and support. I would also like to thank my committee, Professor David Anderson and Professor Chris Rozell for their help in suggestions for my thesis. This work would not have been possible without my fellow lab members in ICElab, without whom it would also have been far less fun. Finally, I'd like to thank my parents for their support and understanding throughout this and all of my endeavours.

TABLE OF CONTENTS

LIST OF FIGURES

SUMMARY

Analog circuit technology is of vital importance in today's world of electronic design. Increasing prevalence of mobile electronics necessitates the search for solutions which offer high performance given tight constraints on power and chip area. Field programmable arrays utilizing floating-gate technology are one possible solution to analog design. It offers the advantages of analog processing with the additional advantage of reconfigurability, giving the designer the ability to test new analog designs without costly and time-consuming fabrication and test cycles.

In this work, a new interface for FPAA's is demonstrated called Sim2spice, with which users can design signal processing systems in Matlab Simulink and compile them to SPICE circuit netlists. These netlists can be further compiled with a tool called GRASPER to a switch list for programming on an FPAA chip. Example library elements are shown, along with some compiled systems such as filters and vector-matrix multipliers.

One particularly compelling application of reconfigurable analog design is the field of neuromorphic circuits, which aims to reproduce the basic functional characteristics of biological neurons and synapses in analog integrated circuit technology. Simulink libraries have been built to allow designers to build neuromorphic systems on several FPAAs that have been developed expressly for the purpose of building neurons and connecting them in networks with synapses. Several possible dynamically learning synapses have also been explored.

CHAPTER 1

INTRODUCTION

As integrated circuit design continues to evolve towards ever faster and smaller chips, digital processing technology generally gets the spotlight. However, the technology in the vast array of cell phones, smart phones, and handheld gaming systems is made possible not only by advancing digital technology, but increasingly on integrated analog functionality. As mobile technology becomes increasingly widespread, power requirements for processing are becoming vitally important. Designers seeking to get the most computing resource out of tight constraints on power turn to analog, as it has a proven track record of outperforming digital technology in terms of processing operations for given power constraints, as Gene's Law shows in Figure 1.

The CADSP, or Cooperative Analog-Digital Signal Processing group at Georgia Tech seeks to achieve the best of both worlds; to bring together the strongest aspects of analog and digital technology and create new technologies that can take power and space-efficient computing to the next level. The Reconfigurable Analog Signal Processor, or RASP, and each iteration of field programmable analog arrays that uses the RASP framework, is an attempt at creating a new type of reconfigurable system that brings together the benefits of analog such as power and space efficiency with the ease of reconfigurability and programmability of the well-known FPGA and DSP type digital devices.

One particularly compelling new application of reconfigurable analog technology is the relatively new field of neuromorphic circuits. These are circuits built to emulate and model the functionality of neurons and synapses. The possible applications of such technology range from interface with live neurons, to new types of machine learning algorithms utilizing learning rules based on local interactions between neurons and synapses. This kind of processing could advance technology in an entirely new direction with a whole new paradigm of computation.

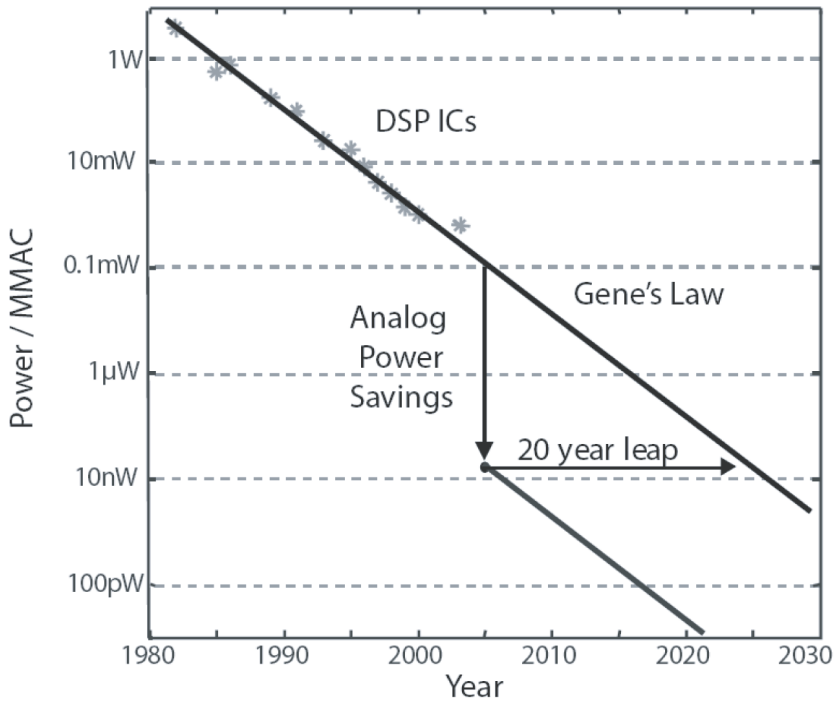


Figure 1. Gene's law. There is a consistent 20 year leap in performance of analog over digital for given power usage. Figure courtesy of Gene Franz [1].

The main goal of this thesis is to demonstrate a new general framework for users to develop signal processing systems on the FPAA. Sim2spice is a tool that allows users to construct block diagrams of signal processing designs in Matlab Simulink, and subsequently compile them to a SPICE netlist and, if desired, to a switch list for immediate implementation on an FPAA chip. This framework was tested on several circuits, and specifically applied to several neuromorphic circuit designs, such as neurons and static and adaptive synapses.

In Chapter 2, I explain the basics of floating gate technology, the applications of floating gate design, and how floating gate transistors can be arranged and individually programmed in an array.

In Chapter 3, I explain the advantages of FPAA chips as signal processing platforms in relation to FPGAs and traditional analog design. I show several different iterations of FPAAs developed within the CADSP group, including FPAAs specifically targetting the

neuromorphic circuit space.

Chapter 4 explores the Sim2spice tool and Simulink libraries built up within the group. Several examples are shown to demonstrate the compilation process, including a simple low-pass filter, and several applications of vector-matrix multiplier circuits.

In Chapter 5, I show how Sim2spice has been applied to the implementation of neuromorphic circuits, such as neurons coupled by static and dynamic symapses. I make mention of the libraries that have been developed containing a wide array of neuromorphic components, and make an attempt at developing a system capable of implementing spike timing dependent plasticity.

Finally, in Chapter 6 I summarize what has been accomplished, my own personal contributions, and the possible avenues for future research of these topics.

CHAPTER 2

FLOATING-GATE ELEMENTS

2.1 Floating Gate Transistors

Floating gate transistors are transistors whose gate is surrounded by insulator with no DC path to ground. It thus retains its charge for very long periods of time, and is often used as a memory element for this reason [2], in applications such as flash memory. When used as an analog circuit element, applications can extend to simple and effective bias generation, multiplier weights for vector-matrix multipliers [3] or synapses [4]. Programming and modifying the floating gate charge results in a shift of the threshold voltage of the transistor, allowing precision in canceling offsets between components and providing biases.

The floating gate thus modulates the current in the channel, and input voltages are coupled to the floating gate by way of coupling capacitors, as shown in Figure 2. The current through the transistor is a function of the floating gate voltage and the drain and source voltages as

$$I_D = I_0 e^{\frac{\kappa V_{fg} - V_S}{U_T}} e^{\frac{V_D}{V_A}} \tag{1}$$

where I_D is the drain voltage, U_T is the termal voltage, and V_A is the early voltage. The floating gate voltage, V_{fg}, is given by

$$V_{fg} = \frac{C_C V_C + C_{tun} V_{tun} + Q}{C_T} \tag{2}$$

where Q is the charge on the floating gate, C_C and C_{tun} are the coupling capacitors for injection and tunneling, respectively, and C_T is the total capacitance. These equations assume the transistor is in saturation. The voltages are referenced to the bulk. If the tunneling voltage V_{tun} equals zero, equation 2 reduces to

$$V_{fg} = V_g \frac{C_C}{C_T} + V_{offset} \tag{3}$$

where V_{offset} is now a voltage offset determined by the charge on the floating gate.

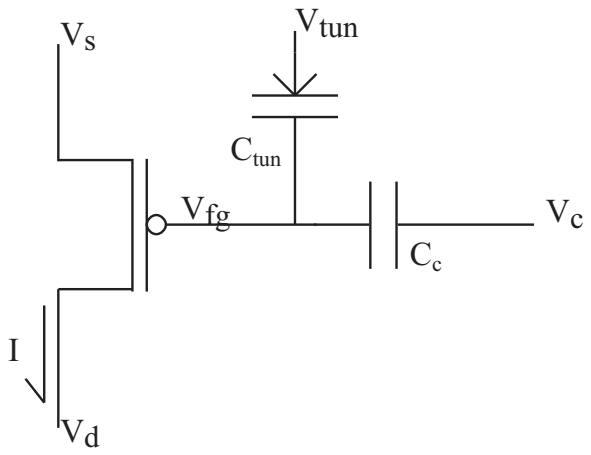


Figure 2. Floating gate circuit schematic. Figure courtesy of Chris Twigg [5].

2.2 Floating Gate Programming

The floating gate charge is modified via the coupling capacitors by two methods; hot-electron injection increases the stored charge, and Fowler-Nordheim tunneling is used to decrease the charge on the gate [7].

Normally, electrons see a large barrier across the oxide in a floating-gate MOSFET transistor. Fowler-Nordheim tunneling is the process of bending the electron bands to such an extend that the barrier across the oxide is easilly overcome by electrons, as shown in Figure 3. This is done by applying a large potential across the tunneling capacitor. This decreases the negative charge on the floating gate, and increases V_{offset}.

Electrons are added to the floating gate by hot-electron injection. A large source-drain voltage is applied to the pFET, as showin in Figure 4. A channel is created in the device by applying a high potential to the gate. Minority carriers are accelerated through the channel as a result of the high field created by the source-drain voltage. When they reach the drain, they collide and create high-eneregy electron-hole pairs. Some of the electrons have enough energy that they penetrate the oxide barrier and move onto the floating gate, increasing the net negative floating gate charge.

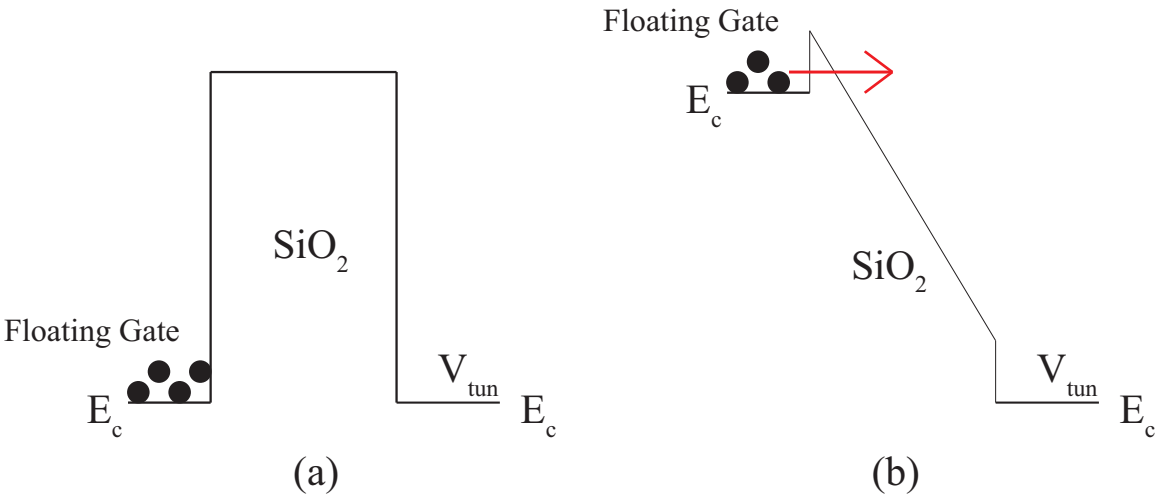


Figure 3. Fowler-Nordheim tunneling is the process by which charge is removed from the floating node of a floating-gate transistor. Figure courtesy of Dave Abramson [6]

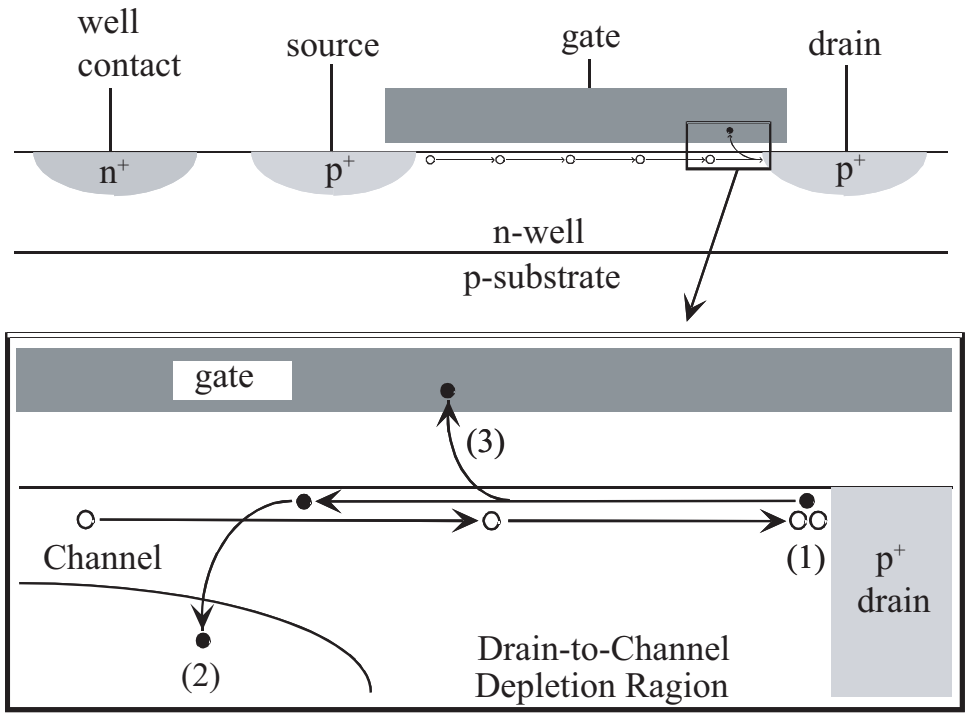


Figure 4. Hot-electron injection is the process by which charge is added to the floating node of a floating-gate transistor. Figure courtesy of Duffy et. al. [8].

2.3 Floating Gate Arrays

Floating gate transistors can be effectively tiled into arrays where the charge on a single isolated floating gate switch can be selectively programmed [9]. It is important for switches to be isolated during programming, so that injecting charge on on switch's floating gate does not affect the charge on any of the others. This is handled by on-chip mux circuits that select specific gate and drain control lines. By applying a drain/gate voltage combination that leads to injection, one particular device will be selected for injection and not any others, if the voltages are only applied to one drain line and one gate line. Tunneling is used as a universal erase step, as the tunneling voltage conditions are applied to all devices simultaneously to restore the charges on their floating gate nodes to the same low level. Tunneling conditions are set by only one parameter, the field across the tunneling MOS capacitor, whereas injection requires the correcct drain voltage and gate voltage. Figure 5 shows an array of floating gates and the drain and gate programming control lines.

In this array, the switches on the switch matrix are directly programmed, which means that when the device operating in a programmed circuit in run mode is the same one whose floating gate is being injected and tunneled. The disadvantages of this approach is that the switch has to be disconnected completely from the any circuit when it is programmed, as it has to be completely isolated. This leads to a higher parasitic capacitance during runtime, as there are more switches in the signal path. Indirect programming is the method used in newer versions of the FPAA to overcome this problem [10]. In an indirect programming scheme, the transistor whose floating gate charge is being modified is not the same one that is actually connected to other circuit elements in run mode. The two transistors share the floating gate, allowing one to be connected to other circuit elements on the FPAA, while the other is reserved purely for programming by injection.

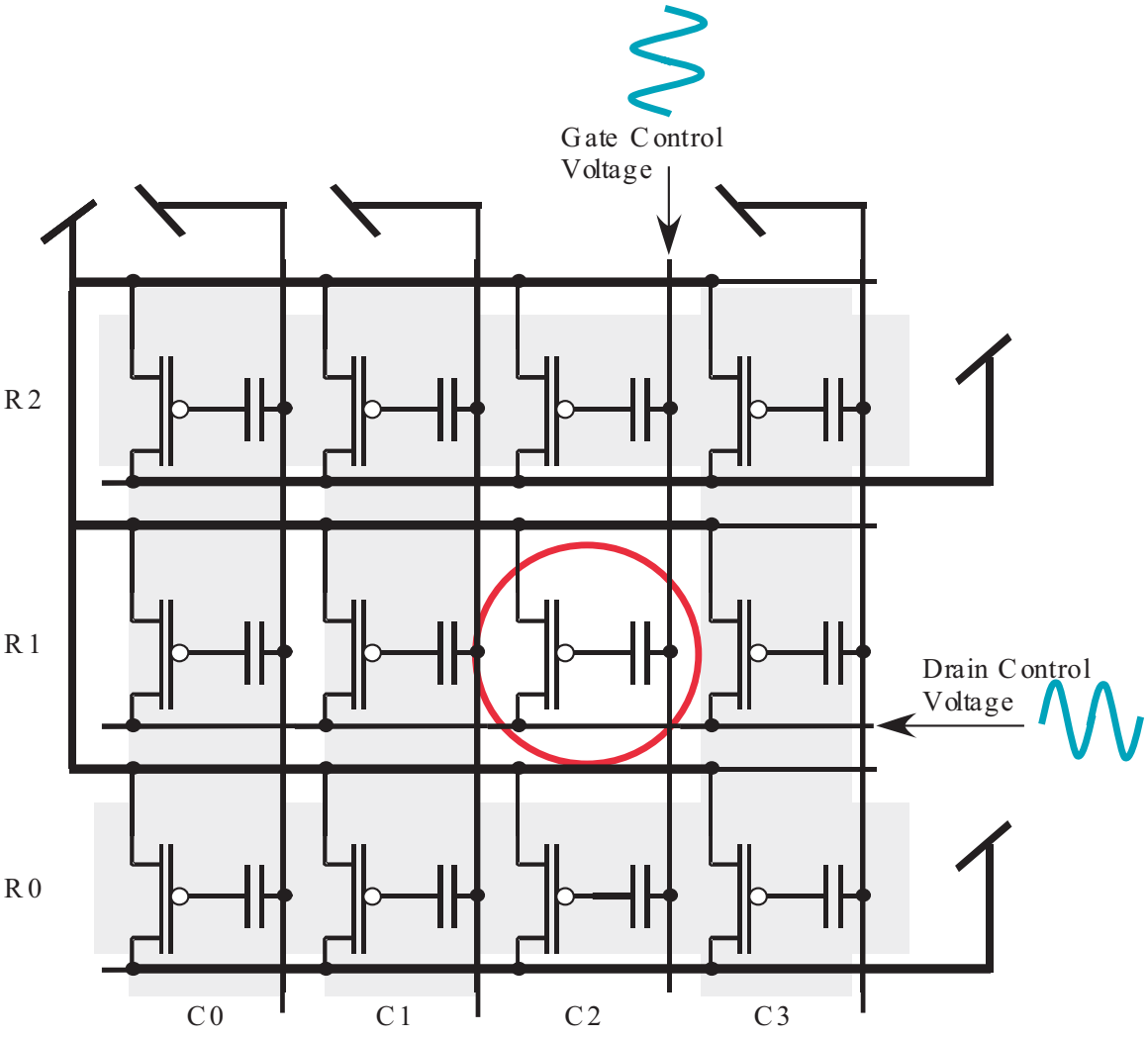


Figure 5. Floating-gate transistor array isolation. Injection requires setting two control lines, drain and gate, leading to isolated programming of floating gate switches. Figure courtesy of Smith et. al. [9].

CHAPTER 3

FIELD-PROGRAMMABLE ANALOG ARRAYS

3.1 The FPAA Advantage

Field programmable analog arrays, or FPAA's, are reconfigurable VLSI analog integrated circuits providing the user with the ability to implement different analog signal processing systems using the same chip simply be reprogramming connection switches in the switch matrices. The desired functionality is similar to FPGA's, except allowing the user to experience the advantages of analog design, such as lower power and higher density for circuits adaptable to analog hardware.

The main advantage of FPAA's over custom analog hardware is reconfigurability. As Figure 6 shows, in a traditional custom analog design cycle, the system is designed, optionally simulated, and must be fabricated and tested before each revision. Fabrication is necessary at each successive iteration of the design, resulting in months of costly lead time between implementations of changes to an analog system. The FPAA allows the entire system to be designed, synthesized, and tested without any new fabrication being necessary. A custom analog IC may be fabricated once the circuit has been improved and modified on the FPAA to a desired level of performance.

3.2 FPAA Architecture

The FPAA's developed in this group, called RASP for Reconfigurable Analog Signal Processor, utilize a switch matrix of floating gate transistors, linking analog components in configurable analog blocks (CABs) [11]. The large switch matrices possible with floating gate FPAA's results from the basic switch element being a floating gate transistor; in traditional approaches, a switch element might be a transmission gate, taking up far more chip area for the same purpose.

One additional advantage of floating gate transistors as switch elements in the FPAA is

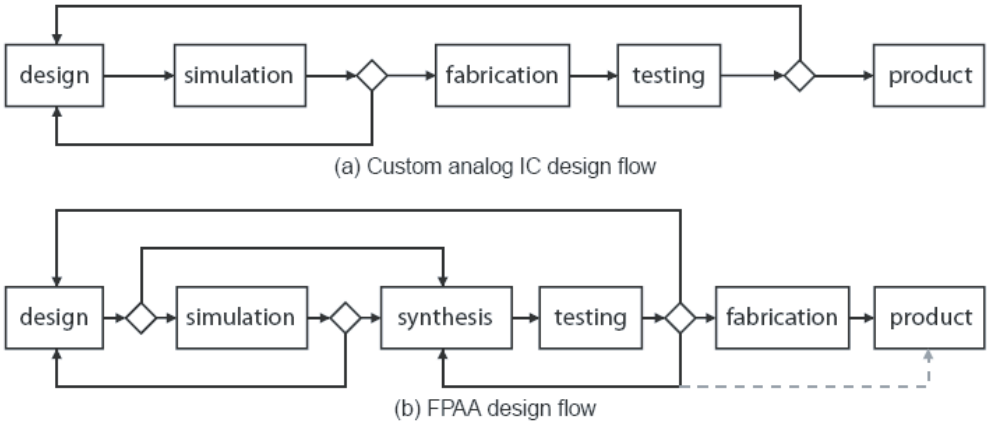


Figure 6. FPAA design flow. (a) Traditional analog design flow, requiring fabrication with each design cycle. (b) FPAA design cycle, with unlimited redesigns and tests before custom IC fabrication. Figure courtesy of Dave Abramson [6]

that they may also be used as computational elements in circuits where this is appropriate, such as vector-matrix mulitpliers [3].

3.3 RASP: Reconfigurable Analog Signal Processor

The RASP series of FPAA's are the focus of the main FPAA work in the CADSP group. One fundamental system shared by all recent RASP chips, such as the RASP 2.8 and 2.9 series, is the onchip programming infrastructure [12]. The main advantage of having the programming system incorporated in the FPAA chip is speed; the main speed increase is due to the on-chip ADC measurement with I to V conversion for measuring currents during programming. Figure 7 shows the on-chip programming infrastructure.

With the switch matrix and programming framework in place, the RASP FPAA infrastructure can be applied to specific applications simply by changing the circuits contained within the CABs. The newest RASP 2.9 FPAAs include numerous variations on the generic chip, which for the most part all build off of the same switch matrix and programming infrastructure.

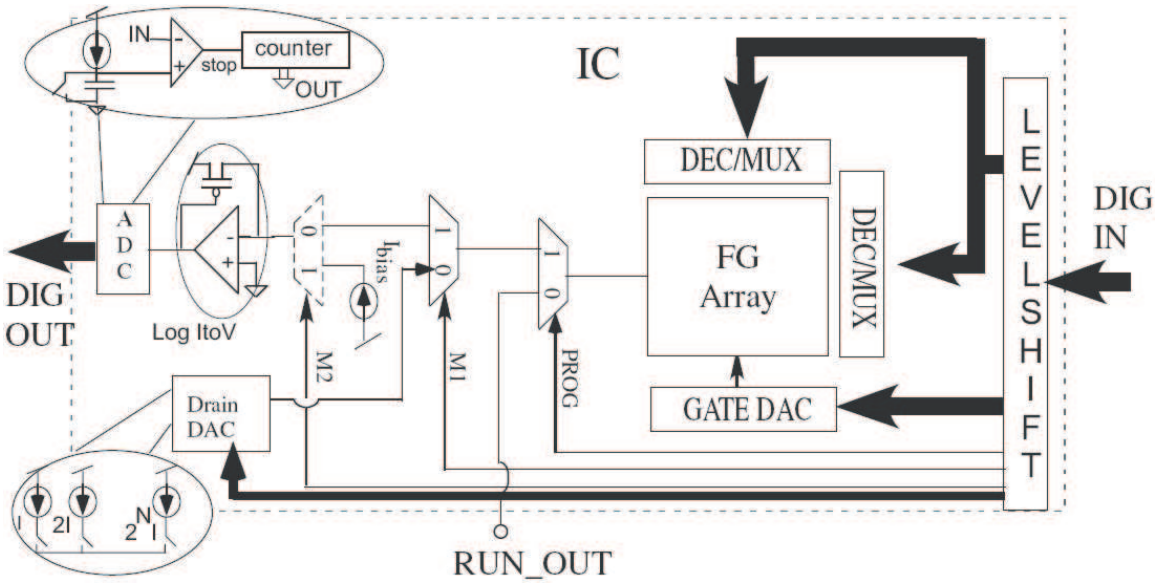


Figure 7. FPAA on-chip programming infrastructure. Figure courtesy of Arindam Basu.

3.3.1 RASP 2.8a

The RASP 2.8a was the general-purpose FPAA of the RASP 2.8 series of chips developed within the CADSP lab [13]. It was fabricated on a 0.35 micrometer TSMC CMOS process. It is composed of 8 rows by 4 columns of CABs, each with associated floating gate switch matrix fabric. The 2.8a was the first FPAA to have localized routing, such as nearest neighbor horizontal and vertical routing lines. Using local routing lines in programmed circuits can drastically reduce the amount of parasitic capacitance, as the routing lines are the primary source of such parasitic capacitance in FPAA circuits. A layout of the 2.8a can be seen in Figure 8. The CABs in the RASP 2.8a contain often used analog elements, such as 500fF capacitors, nFET and pFET transistors, operational transconductance amplifiers, and buffers.

3.3.2 RASP 2.9a

The RASP 2.9a is the newest general-purpose FPAA. It is essentially the same chip as the 2.8a, except made far larger. The 2.9a has 12 rows and 6 columns of CABs, allowing larger circuits to be developed, such as very large vector-matrix multipliers that use the large

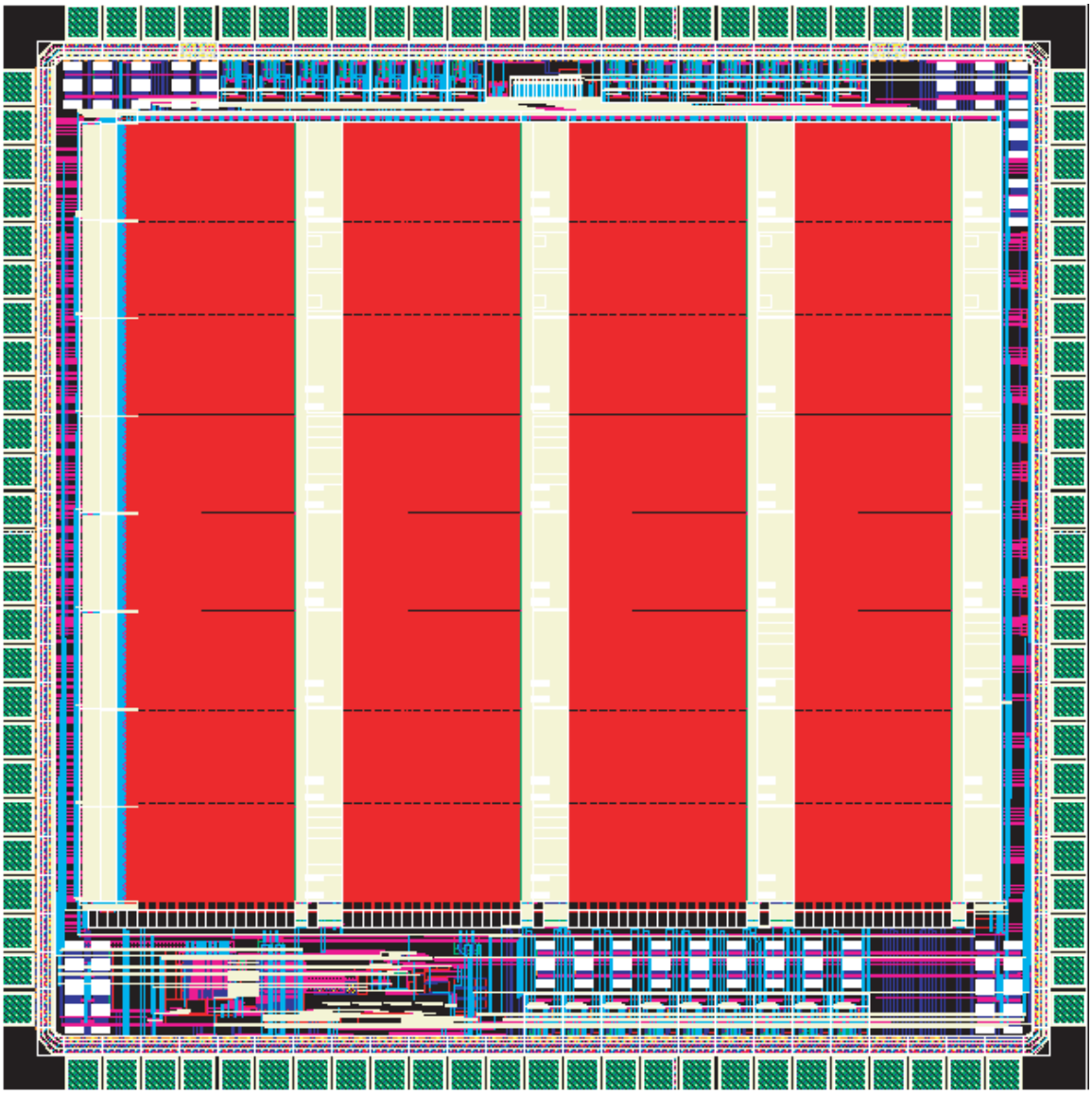


Figure 8. RASP 2.8a layout.

amount of switches in the switch matrix. It also has additional CAB components, such OTAs with programmable floating gate inputs, signal-by-signal multipliers, and pinned out floating gates. Figure 9 shows a layout of the RASP 2.9a FPAA.

3.3.3 RASP 2.9c

The RASP 2.9c FPAA, or Bio FPAA, has a routing infrastructure identical to the RASP 2.9a, but some of the CAB elements are neuromorphic devices. There are no adaptive elements on the 2.9c, but there are neurons and static synapses. Figure 10 shows the RASP 2.9c chip layout.

3.3.4 RASP 2.9f

The RASP 2.9f FPAA was designed specifically to implement timing-based adaptive synaptic learning rules, such as STDP. The CAB contains components such as current starved inverters for delays, some basic analog elements such as capacitors and OTA's, and adaptive synapse elements using digital control signals to control injection and tunneling in run mode. Figure 11 shows the elements of one of the adaptive CABs on the 2.9f FPAA.

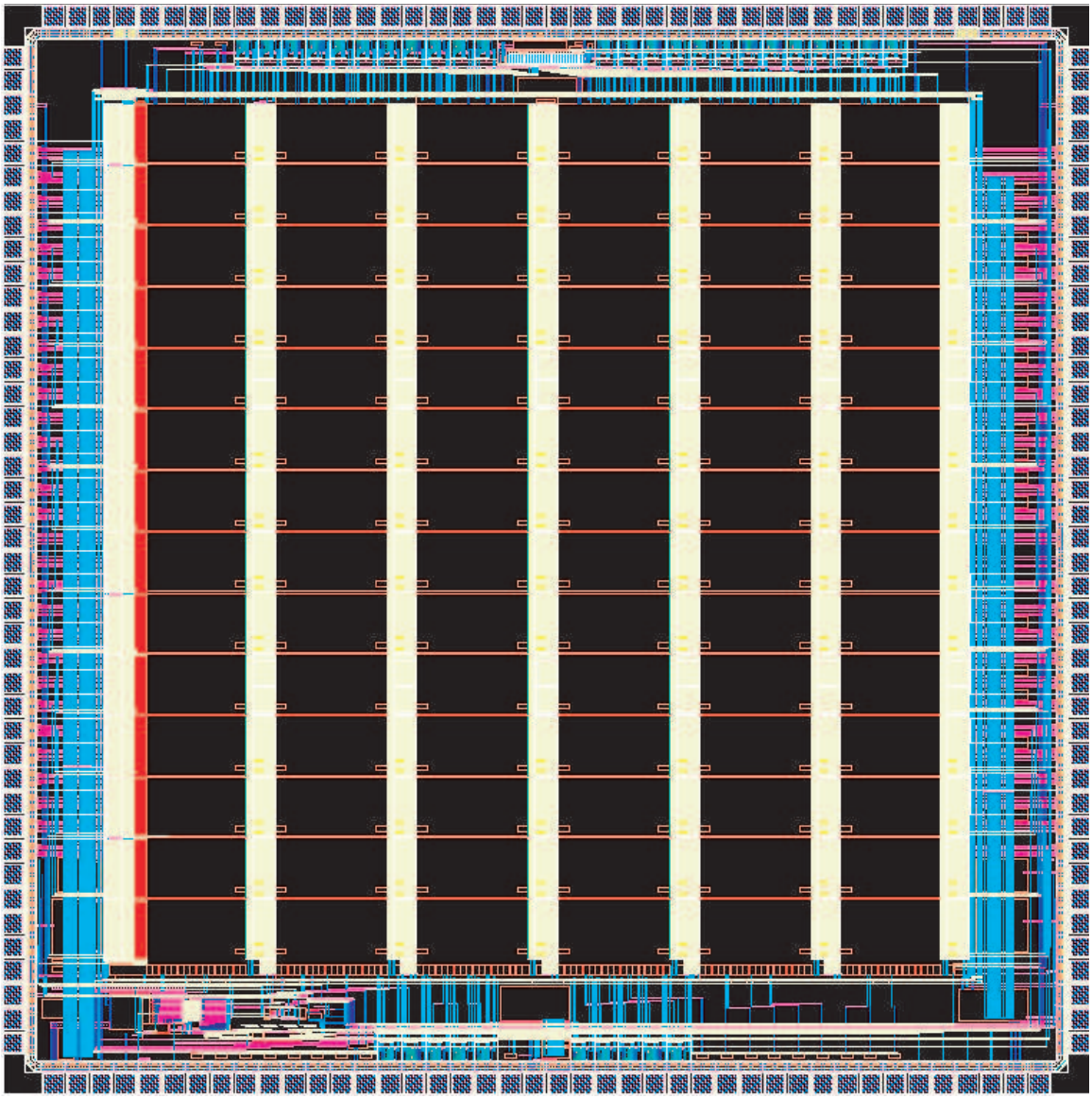


Figure 9. RASP 2.9a layout.

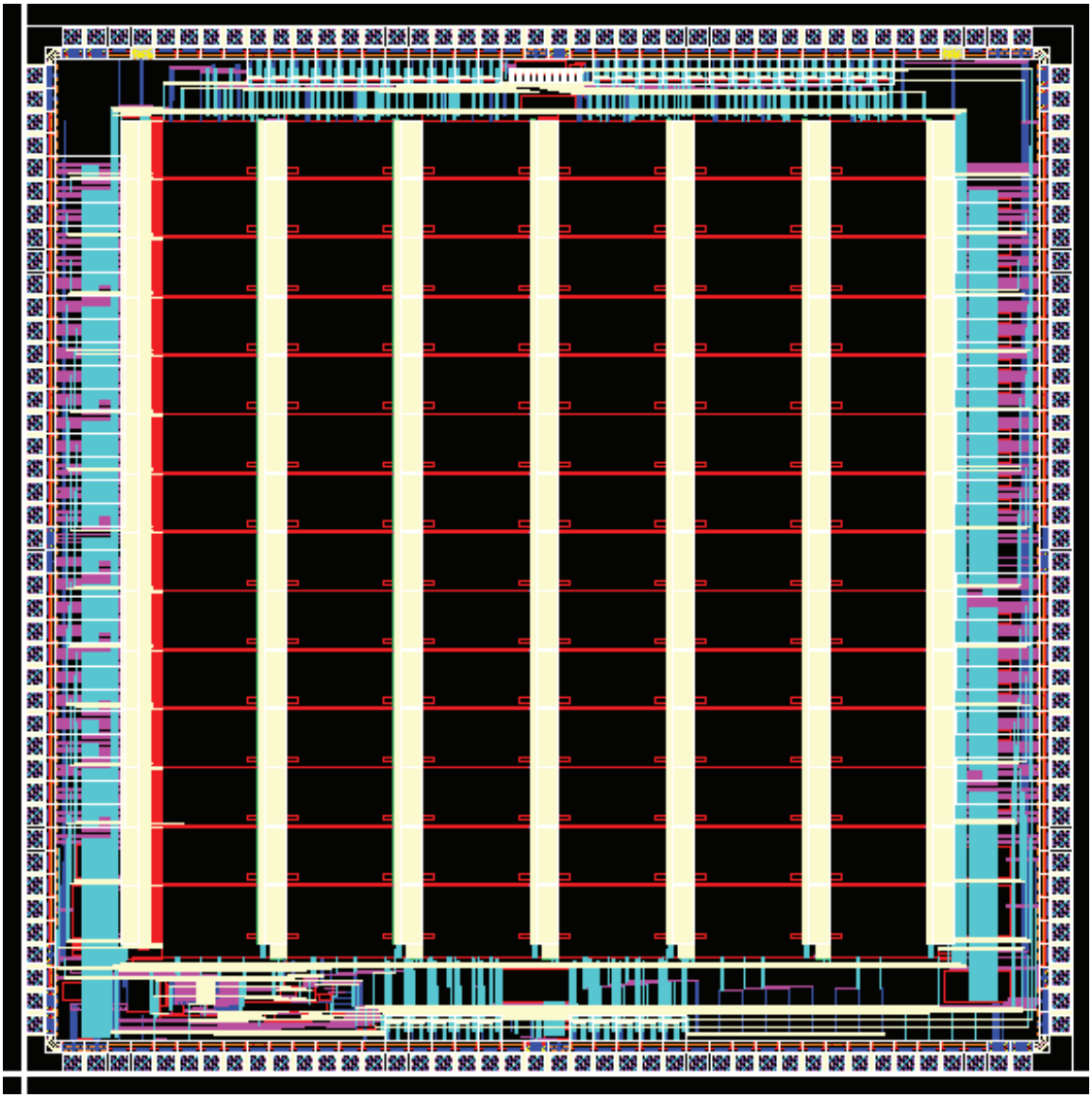


Figure 10. RASP 2.9c layout.

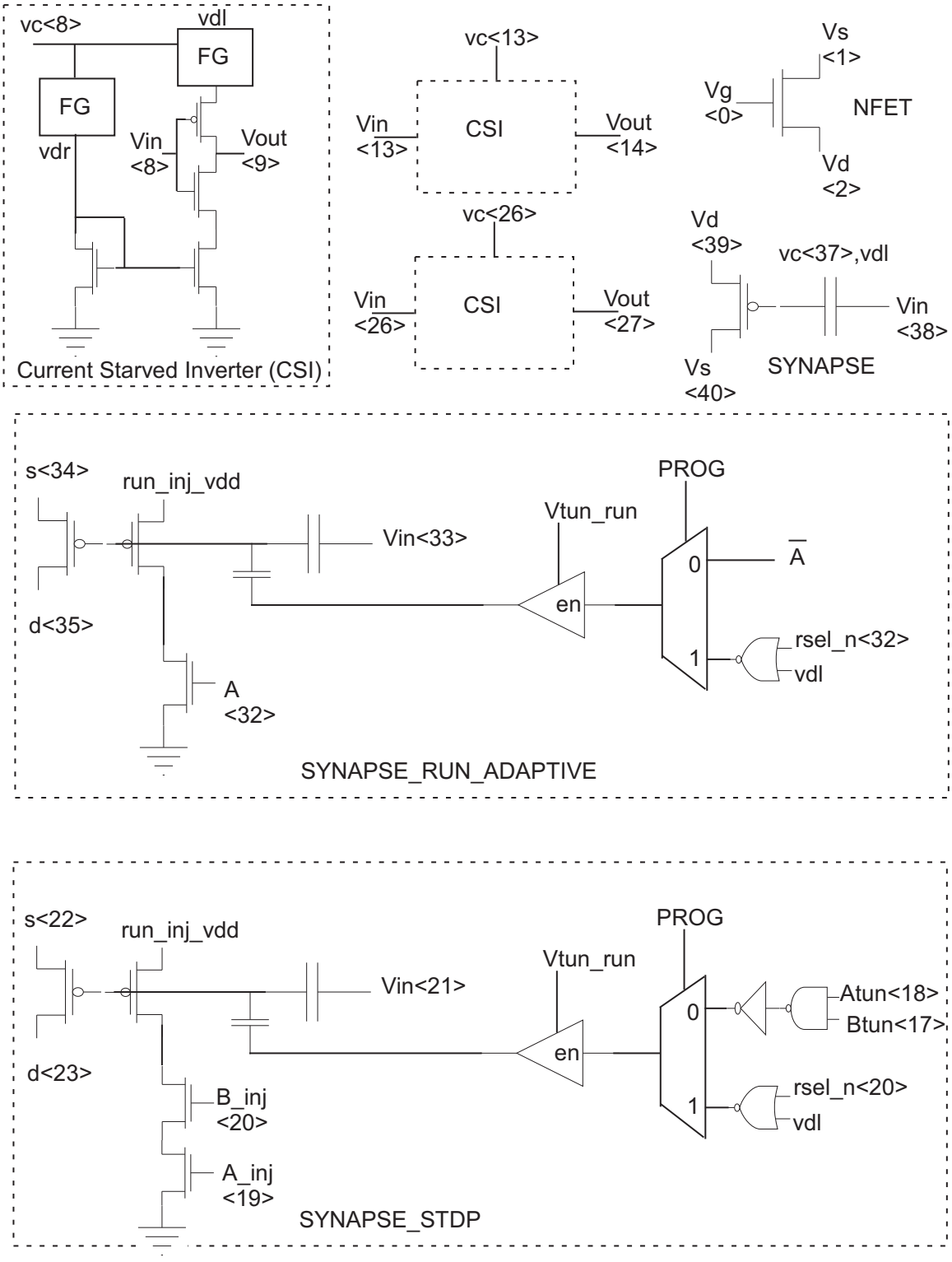


Figure 11. The CABs on the RASP 2.9f FPAA have adaptive components. The digital logic control signals and delay elements are infrastructure for spike timing dependent plasticity (STDP). Figure courtesy of Arindam Basu.

CHAPTER 4

SIM2SPICE AND TOOL FLOW

Simulink is a subset of the Mathworks Matlab software tool. It provides an intuitive visual interface for engineers to design and simulate signal processing systems. The user has the ability to connect signal processing elements together as blocks linked by wires. The system can then be simulated in real time. There are several existing prototype software tools that allow users to compile Simulink models to digital hardware on an FPGA, such as [14] and [15]. As of this research, there is no equivalent tool for analog hardware. Sim2spice is designed as a tool to allow engineers to compile signal processing systems directly from Simulink block diagrams to SPICE circuit netlist, which can then be further compiled to a list of switches to be programmed on the FPAA with the help of a code, GRASPER, written by a colleague [16]. With this software tool, a user can compile and implement a signal processing system designed using Simulink directly in analog hardware [17]. The general concept of the Sim2spice tool functionality is shown in Figure 12. The complete tool flow of which Sim2spice is a part, showing the process of compiling from Simulink model to FPAA switch list, is shown in Figure 13.

4.1 Sim2spice Code

The Sim2spice code is actually composed of two components; the Simulink model parser, which loads and parses the Simulink .mdl file into a Matlab structure, and the SPICE netlist generator, which uses the generated structure to create a SPICE netlist for the model. Figure 14 shows a block diagram of the major Sim2spice code components.

4.1.1 Simulink Model Parser

The Sim2spice parser is called from Matlab. It is a custom code written in python that has been packaged as a Windows executable program. The input to the script is a Simulink model (.mdl) file, and the output is a Matlab structure containing a list of blocks, connection

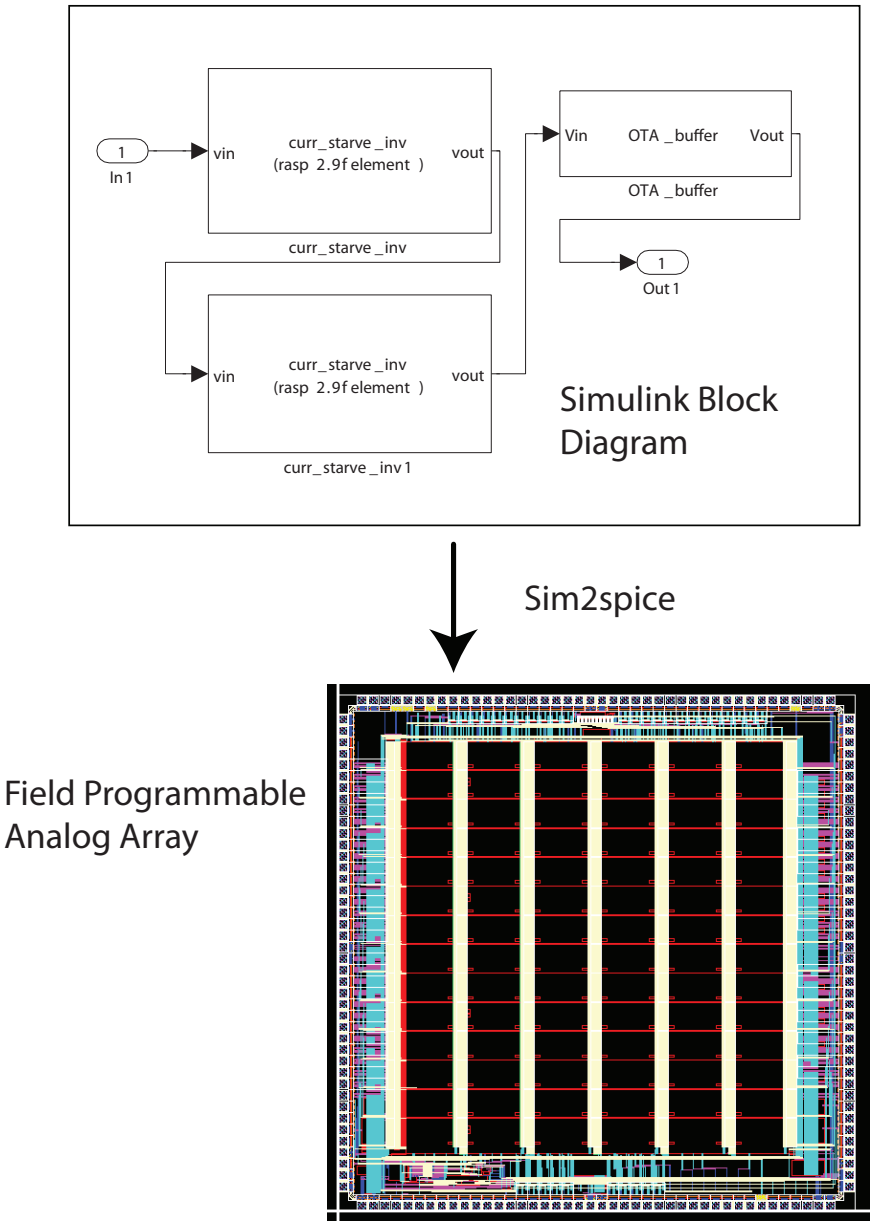


Figure 12. Sim2spice is a software tool for compiling systems in Simulink block diagram form to FPAA.

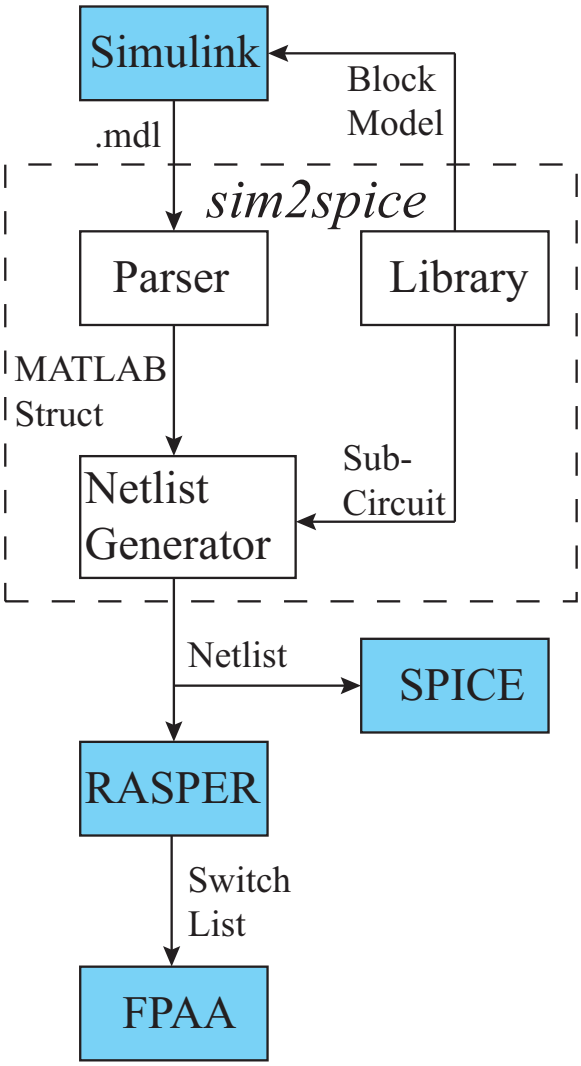


Figure 13. The complete tool set is comprised of Sim2spice, which converts a Simulink design to a SPICE netlist, and GRASPER, which converts a SPICE netlist to a set of switches for programming on the FPAA.

lines, and all of the associated block parameter types and values. Python was the chosen language for development of the parser due to the ease of parsing text with built in libraries such as PyParsing [18].

4.1.2 SPICE Netlist Generator

The netlist generator converts the structure obtained in the previous step to a SPICE netlist, utilizing the SPICE circuit elements contained in the Sim2sice circuit library associated with each Simulink block in the structure.

The generator first reads a full list of all block types from the component libary, then reads a description file for each block type. The description file lists attributes and changeable parameters of the block type, such as input and output port sizes and other user-defined parameters. Once these files have been read, the information about the library is passed to the parser, which is then called to parse the actual model file. It uses the information about block types and their possible parameters to read the .mdl file and look for the appropriate blocks and parameter values, while building a Matlab structure.

The structure that is the output from the parser contains an array of blocks and an array of lines, which are connections between outputs of one block and inputs of another. The netlist generator must then make several passes over this array and find common nets between blocks and giving them unique identifier names. There are several operations that must be performed in this process that make this a non-trivial problem. Some blocks have input and output ports that, in Simulink, are different ports, but as a circuit they correspond to the same net; for instance, if a circuit has a voltage input and current output on the same net. These nets must be propagated from input to output of that block and to all other inputs and outputs of any blocks that may be connected to that block. In addition, all of the parts in the Sim2spice library are designed in a vectorized fashion. That is, input and output ports are taken to be arrays of signals, with an array of components connecting them in parallel. The user can define a size attribute for the block in Simulink to give a size for this array. The lines connecting such blocks must be a set of lines in parallel, with the vectorized size

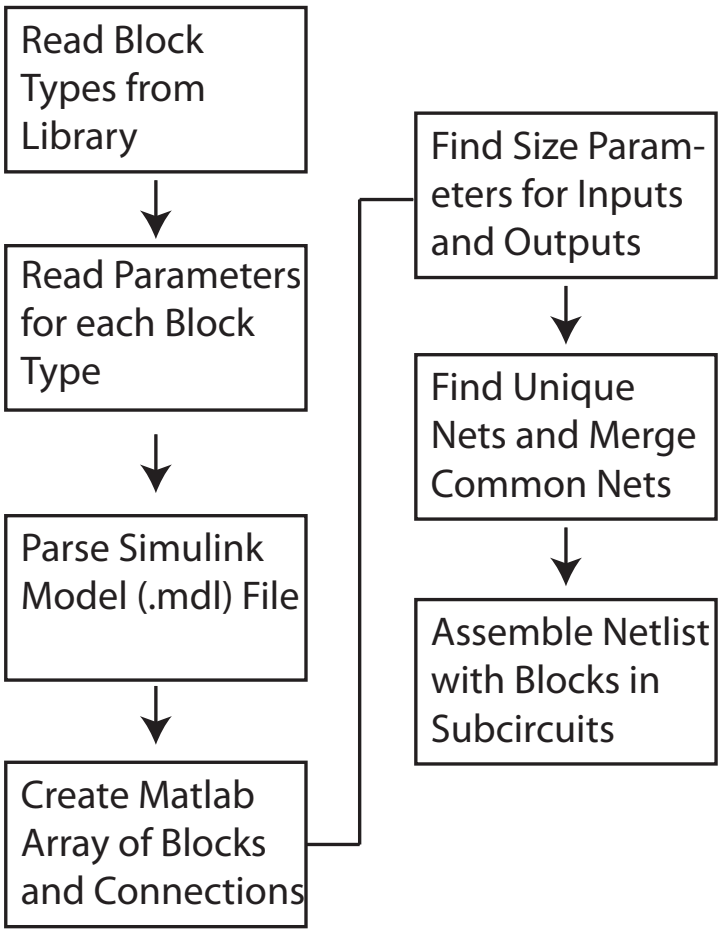


Figure 14. The Sim2spice code involves several steps to organize the design into unique nets, read parameters from the library, and write a complete SPICE netlist.

of the lines matching the input and output port sizes of the blocks.

Once connections have been determined between blocks and unique and common nets have been identified and named, the next step is the actual assembly of the SPICE netlist. The program steps from block to block in the array and calls a user-defined Matlab script, or build file, for each block. The build file takes as input the uniquely determined net names for the inputs and outputs to that block and the parameter values for that block passed to the program from the Simulink model file. The build file then assembles a SPICE netlist for that block given the net names and parameter values. This SPICE netlist, provided as a cell array of strings where each string represents a line of the netlist file, is the output of the

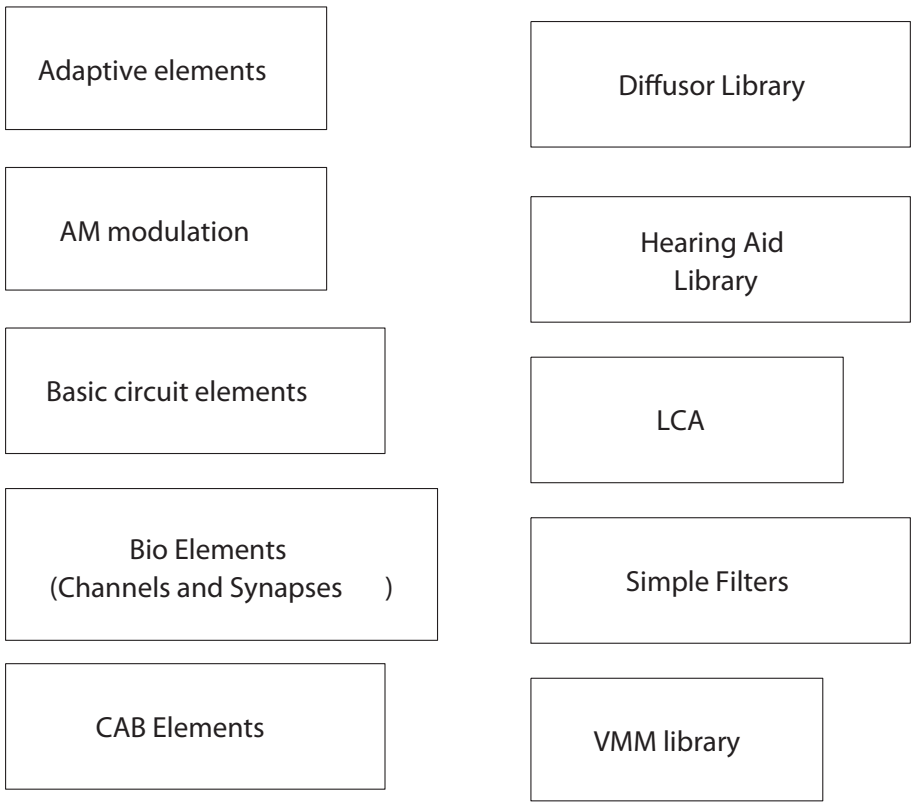


Figure 15. Sim2spice library organization. These libraries are easilly accessible in Simulink, and the blocks contained in any library can be placed in any Simulink model and connected to other blocks.

build file. The netlist generator then combines the SPICE files assembled for each block into one SPICE netlist for the entire circuit representing the model file. The net names are kept unique by encapsulating each block instance in a SPICE subcircuit. The input and output ports in the Simulink model become input and output pins in the SPICE netlist, and they are converted to I/O pins on the FPAA when the netlist is compiled with GRASPER.

4.2 Sim2spice Library

The Sim2spice block library consists of a set of custom level 2 M-file S-function Simulink blocks and their corresponding SPICE circuit netlists. Adding a new block to the library actually requires adding four different independent units. These are the Simulink S-function block, which is the graphical representation of the block in Simulink and allows the user to

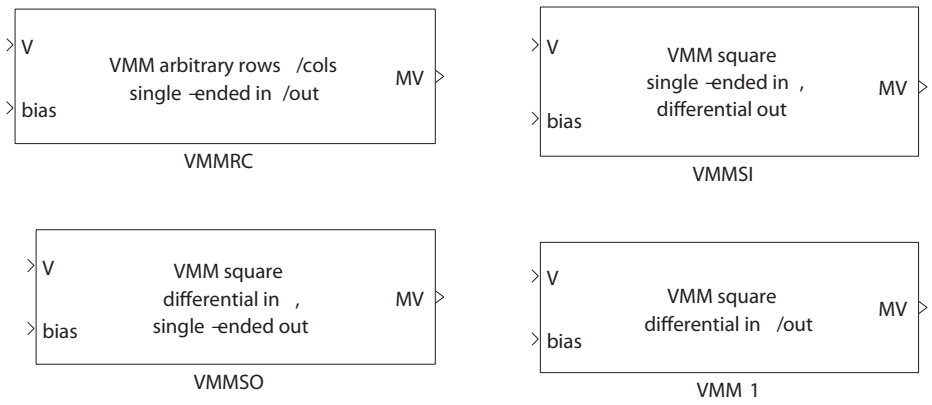


Figure 16. Sim2spice vector-matrix multiplier library.

set parameter values, the associated Matlab script (.m file) which encapsulates the dynamic behavior of the block in Simulink, the Matlab build file which tells the netlist generator how to use block parameter values and input/output ports to assemble the SPICE circuit netlist for the block, and a description file which lists input/output port parameters and other block type attributes and user-changeable parameters for the block, which is used by the parser to determine what parameter values to read from the Simulink model file in which the block is used.

The library now contains a large variety of blocks in 11 different subcategories of functionality, and is growing quickly as users continue to add new blocks [19]. The currently existing categories include basic FPAA CAB elements, simple filters, neuromorphic elements such as neuron and synapse models, vector-matrix multipliers, current-to-voltage and voltage-to-current converters, output buffers, and adaptive elements such as adaptive synapses. Figure 15 shows the organization of the library into categories.

Each block is copied into a new design from the library. At that point, it becomes an instance of that block type. The parameters for that particular instance of the block can be changed by double clicking on the block in the Simulink design. These parameters will then determine the appropriate properties for the corresponding circuit element when the design is compiled to a SPICE netlist, and then to the FPAA.

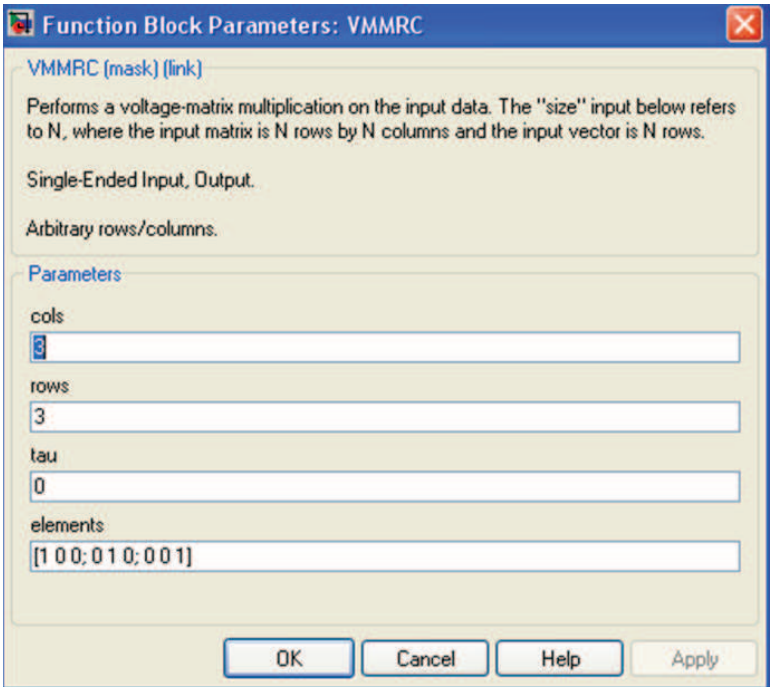


Figure 17. Vector-matrix multiplier parameter dialog box.

4.2.1 Vector-Matrix Mulitpliers

One example library that was one of the first to be developed is the VMM library. The blocks comprising the VMM library are shown in figure 16. Figure 17 shows the dialog box for changing the parameters for a VMM block instance. The circuit netlist primitive in the library for the VMM is shown as a schematic in Figure 18.

4.2.2 Neuromorphic Elements

Several neuromorphic elements have been implemented as Simulink blocks with corresponding SPICE circuit library elements. These parts include sodium and potassium channels for building Hodgkin-Huxley type spiking neurons [20] and various synaptic connections and synapse elements [21]. These parts and some example systems are described in more detail in the next chapter.

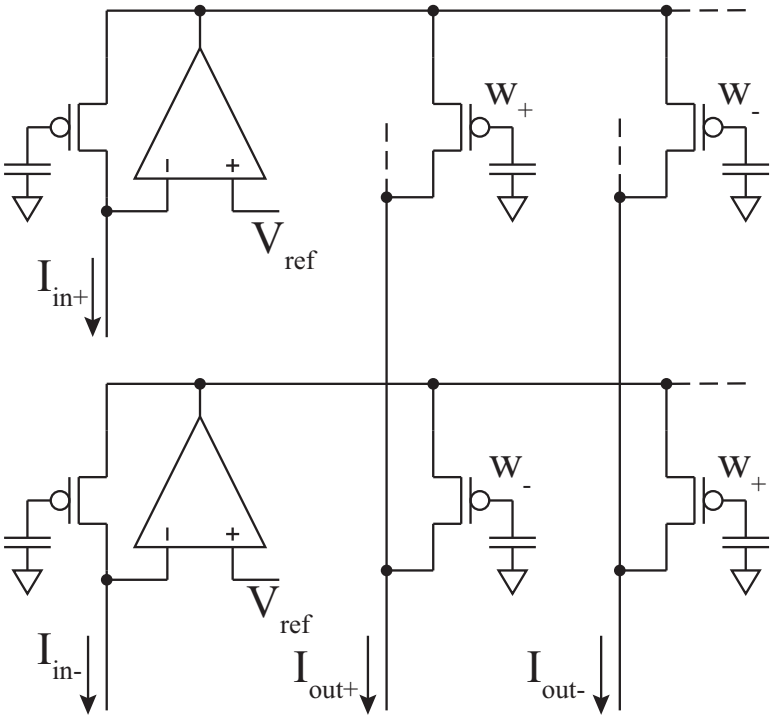


Figure 18. Vector-matrix multiplier circuit schematic. Figure courtesy of Craig Schlottmann.

4.2.3 Adaptive Elements

One FPAA model from the 2.9 series is the 2.9f, or adaptive FPAA. This FPAA contains circuit elements that can act as adaptive synapses, with additional digital logic supplying infrastructure with the goal of implementing spike-timing dependent plasticity (STDP). The implementation of this circuit in Simulink and compilation and testing efforts in FPAA are detailed in the next chapter.

4.3 GRASPER

GRASPER is the latest version of a tool developed by a colleague for the purpose of compiling SPICE circuit netlists to a list of routing switch locations on an FPAA switch matrix [22].

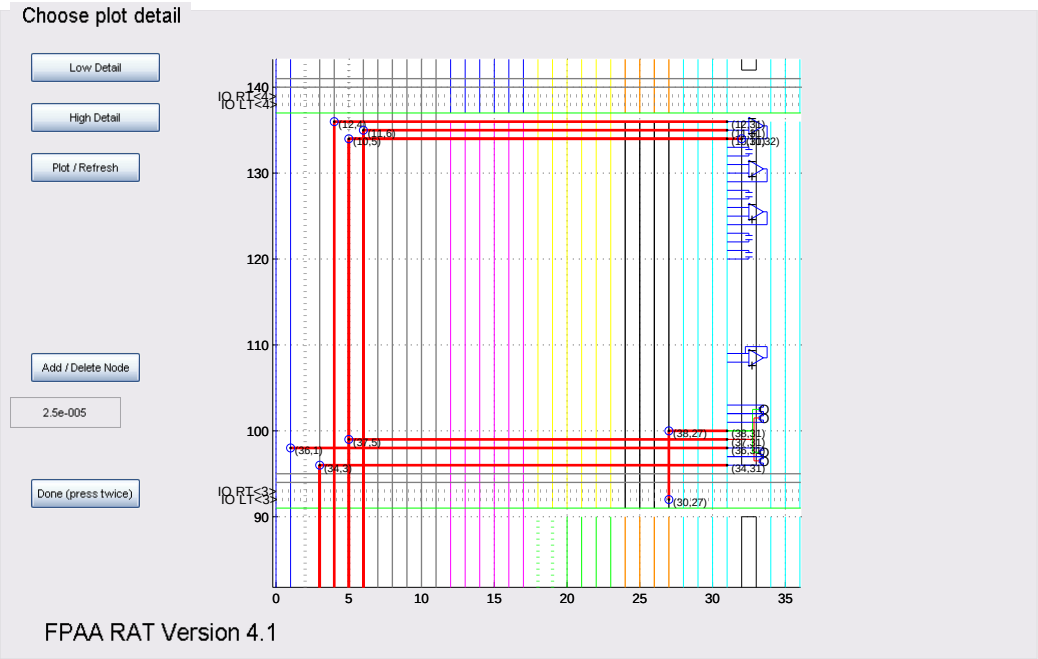


Figure 19. Routing Analysis Tool. This Matlab program allows the user to visually verify and change FPAA switch lists.

4.3.1 Routing Analysis Tool

The Routing Analysis Tool, or RAT, is a tool developed by a colleague for the purpose of displaying a list of FPAA routing switch locations [23]. It is a useful tool for diagnosing errors in small circuits given only a list of switches, for building new circuits in a visual interface, and for making modifications and additions to existing circuits. Figure 19 shows a screen shot of the Matlab RAT interface with a simple example circuit on the RASP 2.8aa chip.

4.4 FPAA Board

The custom-designed FPAA circuit board supplies power, signal interfacing, and other necessary infrastructure for the FPAA chip. The main features of the FPAA board are a 40-channel DAC chip, a 4-channel 12-bit ADC chip, a 12-volt charge pump voltage supply and amplifiers required for the programming circuitry on the FPAA, audio amplifiers, and

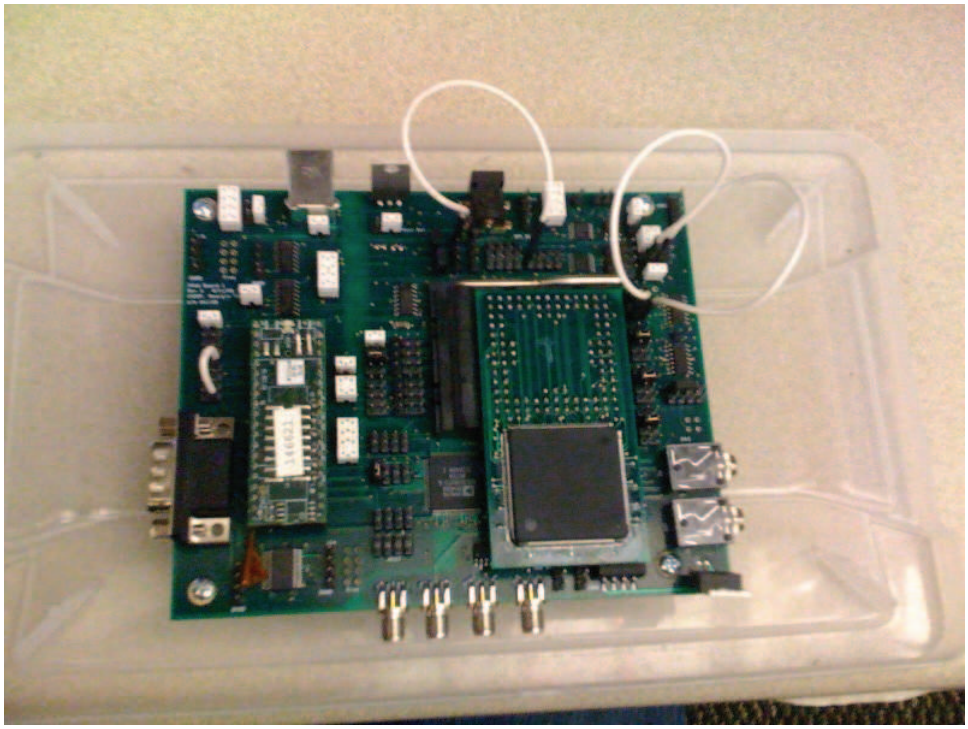


Figure 20. Board supplying infrastructure and interfacing for the FPAA, including a 40-channel DAC chip and audio amplifiers.

a SAM7S microcontroller in a 40-pin SAMDIP package. The microcontroller is the main interface between the computer and the FPAA. DAC, digital I/O, and programming commands are sent to the FPAA through the microcontroller. The microcontroller receives power and commands through the USB port on the board, and the computer is connected to the board with a USB cable. The microcontroller communicates with the other chips on the board over Serial Peripheral Interface (SPI) Bus. The FPAA board is shown in 20.

4.5 Example Systems

4.5.1 Low-Pass Filter

One simple and often-used signal processing system is a filter, such as a low-pass filter. The filter that was implemented here is an OTA-Capacitor filter, utilizing an operational transconductance amplifier and setting the time constant with a capacitor at the output. The output voltage is then passed through an OTA buffer. Figure 21 shows the model as

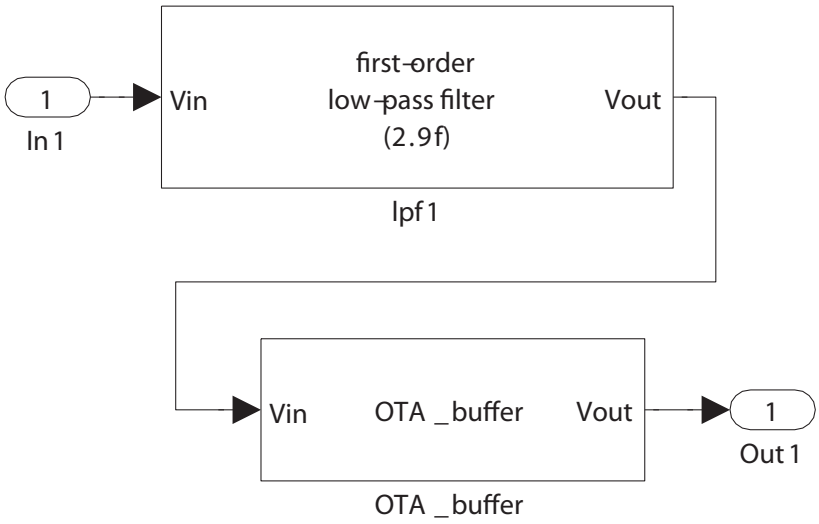


Figure 21. Low Pass Filter: Simulink model

constructed in Simulink. This filter was then compiled using Sim2spice, and resulted in a SPICE netlist. The netlist can be seen in Figure 22.

After construction of the circuit in Simulink, voltage sources were added to the netlist by hand, and it was simulated using WinSPICE, a free SPICE circuit simulator program for Microsoft Windows. Figure 23 shows the SPICE simulation result for the step response for the low-pass filter.

Finally, the generated SPICE netlist was compiled using GRASPER to a list of routing switches for the RASP 2.9f FPAA, as shown in Figure 24. The routing switches were programmed on the FPAA, and the performance of the circuit was verified for a step response. Figure 25 shows the step response for the low-pass filter as implemented on the FPAA. The time constant in the FPAA experiment is different from the ideal SPICE simulation due to the difficulty of calculating the exact capacitance on the FPAA at the output node of the filter; the routing lines add significant coupling capacitance that changes the time constant of the filter.

```
*INPORT In1
*>> pin io_lt 0 net int1net1

.SUBCKT ota_buffersub2 int2net1In int3net1Out
Xota_1 int2net1In int3net1Out int3net1Out OTA PARAMS: Ib=2e-006
.ENDS

Xota_buffersub2 int2net1 int3net1 ota_buffersub2

.SUBCKT first_order_lpf_2_9fsub3 int1net1In int2net1Out
Xota1_1 int1net1In int2net1Out int2net1Out OTA PARAMS: Ib=1e-009
Xc1_1 int2net1Out CAP
Xc2_1 int2net1Out CAP
Xc3_1 int2net1Out CAP
.ENDS

Xfirst_order_lpf_2_9fsub3 int1net1 int2net1 first_order_lpf_2_9fsub3

*OUTPORT Out1*>> pin io_dn 0 net int3net1

*configuration files (can also be entered in the command line)
*>> devicefile rasp2_9f.dev
*>> project work
```

Figure 22. Compiled low-pass filter SPICE netlist.

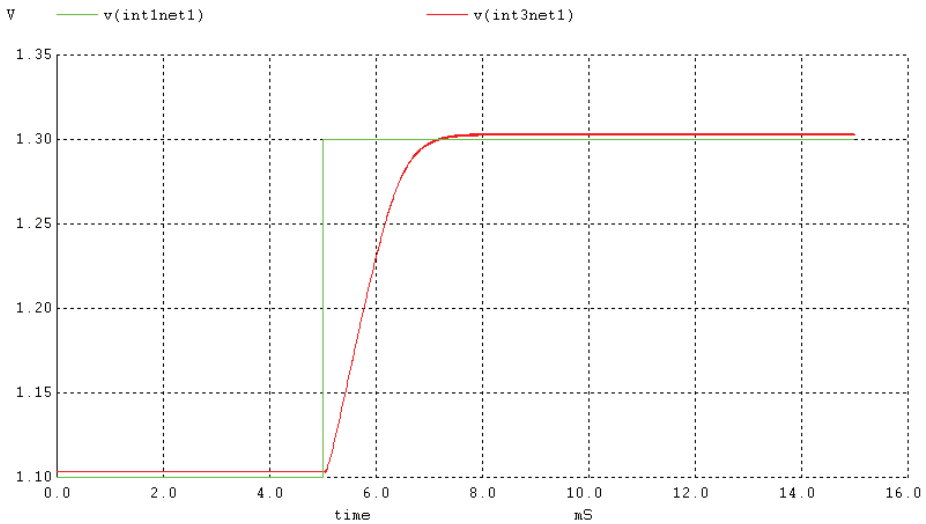


Figure 23. Low Pass Filter: Step response in SPICE simulation.

```
%routing switches
0 14 1.8
41 6 1.8
41 13 1.8
42 14 1.8
43 13 1.8
133 4 1.8
134 6 1.8
135 4 1.8

%configuration switches
133 32 0.001
41 32 1e-09
```

Figure 24. Low pass filter FPAA switch list.

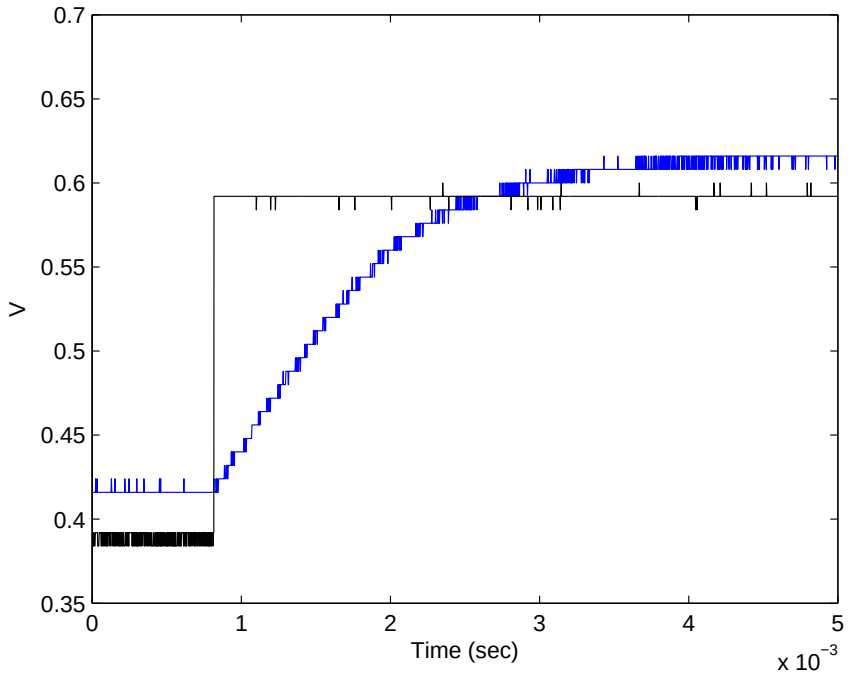
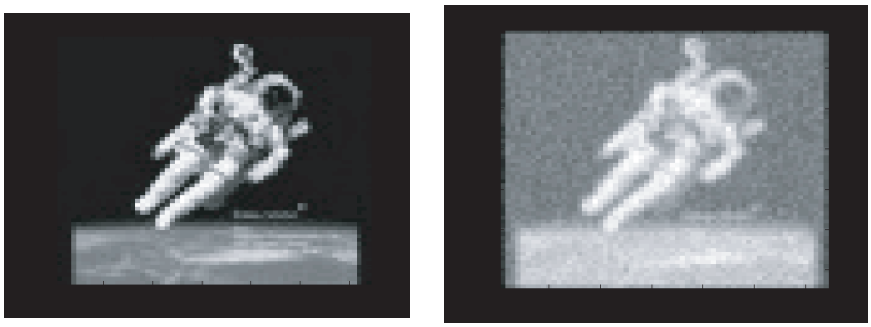


Figure 25. Low Pass Filter: Step response on FPAA



(a) (b)

Figure 26. 2-D Gaussian filtering. (a) Original image. (b) Image after Gaussian filtering on FPAA.

4.5.2 2-D Gaussian Image Filtering

The VMM library was used to produce a two-dimansional Gaussian smoothing image filter. The VMM structure used was a 1x5 VMM with 5 weights, programmed to values representing a Gaussian weight distribution. The VMM was designed in Simulink and the proper weights inputted through the dialog box for the VMM block. The system was then compiled on the RASP 2.8aa FPAA. The input image used was a 64x64 pixel grayscale image. The one-dimensional Gaussian filter as programmed on the FPAA was applied to the first the rows, then the columns of the image, resulting in a two-dimensional smoothing transform. Figure 26 shows the original image and the output image after it has been passed through the Gaussian smoothing filter.

4.5.3 Discrete Cosine Transform

The transformation matrix for a 4x4 discrete cosine transform (DCT) was programmed on the FPAA with the use of Sim2spice. A 4x4 differential input, single-ended output VMM block was instantiated in Simulink and the values filled in for a 4x4 DCT transform. The nanoamp-range input currents were set through very large, 20-megaohm resistors, and the output currents were switched between the four rows with transmission gates found on the FPAA and read through a picoammeter. The DCT transform was done on the FPAA, and the inverse DCT transform was done on the transformed image in Matlab. Figure 27 shows the 64x64 grayscale input and output images for the DCT transform, with the transform

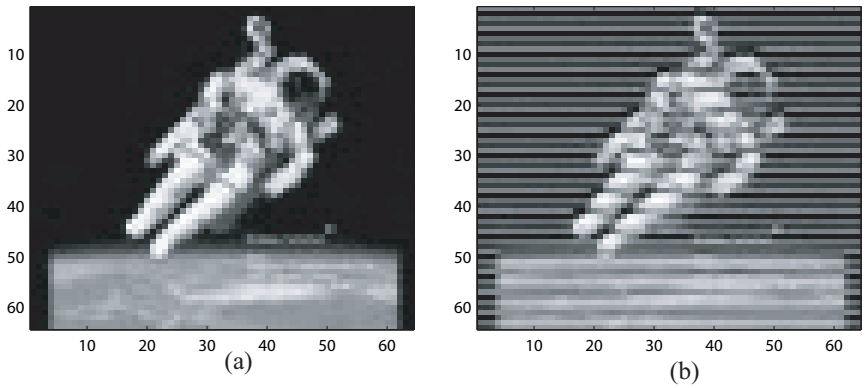


Figure 27. Discrete cosine transform. (a) Original image. (b) Image after DCT transform on FPAA and inverse transform in Matlab.

done on FPAA and the inverse transform on the computer.

CHAPTER 5

NEUROMORPHIC CIRCUITS

Several basic neuromorphic functional elements have been implemented and tested on several different versions of the FPAA and in custom analog hardware. Several models for static and dynamic synapses, acting as connections between neurons, have also been developed. These models have been added to the Simulink library and Sim2spice was used to compile and test them on FPAA. The Simulink library for static neuromorphic elements, such as Hodgkin-Huxley type neurons, integrate and fire neurons, and several different static synapses is shown in Figure 28.

5.1 Neuron Chip

The first neuromorphic element studied was a multiple transistor, analog implementation of a Hodgkin-Huxley spiking neuron model. This kind of model is based on the modeling of individual ion channels in neurons, such as sodium and potassium channels, as transistors based on the similarites of their inherent physical dynamics [24]. The particular model explored is a neuron composed of a single sodium and a single potassium channel. The channels are constructed as models of the movement of ions across the cell membrane, rather than as an attempt to model a specific set of equations such as the accepted Hodgkin-Huxley differential equations [25]. This set of four equations was the first and still most widely accepted model of the spiking dynamics of a neuron. The main disadvantage of such a model is that four coupled differential equations, especially given the multiple time scales of neural dynamics, are difficult to simulate in real time.

One way to achieve real-time simulation of neural systems is by simulating a neuron in analog hardware. There has been a historical trend in attempting to model neural dynamics in analog hardware. Some of the earliest work on modeling neurons as dedicated silicon hardware was by Mahowald and Douglas [26]. Our model takes advantage of the inherent

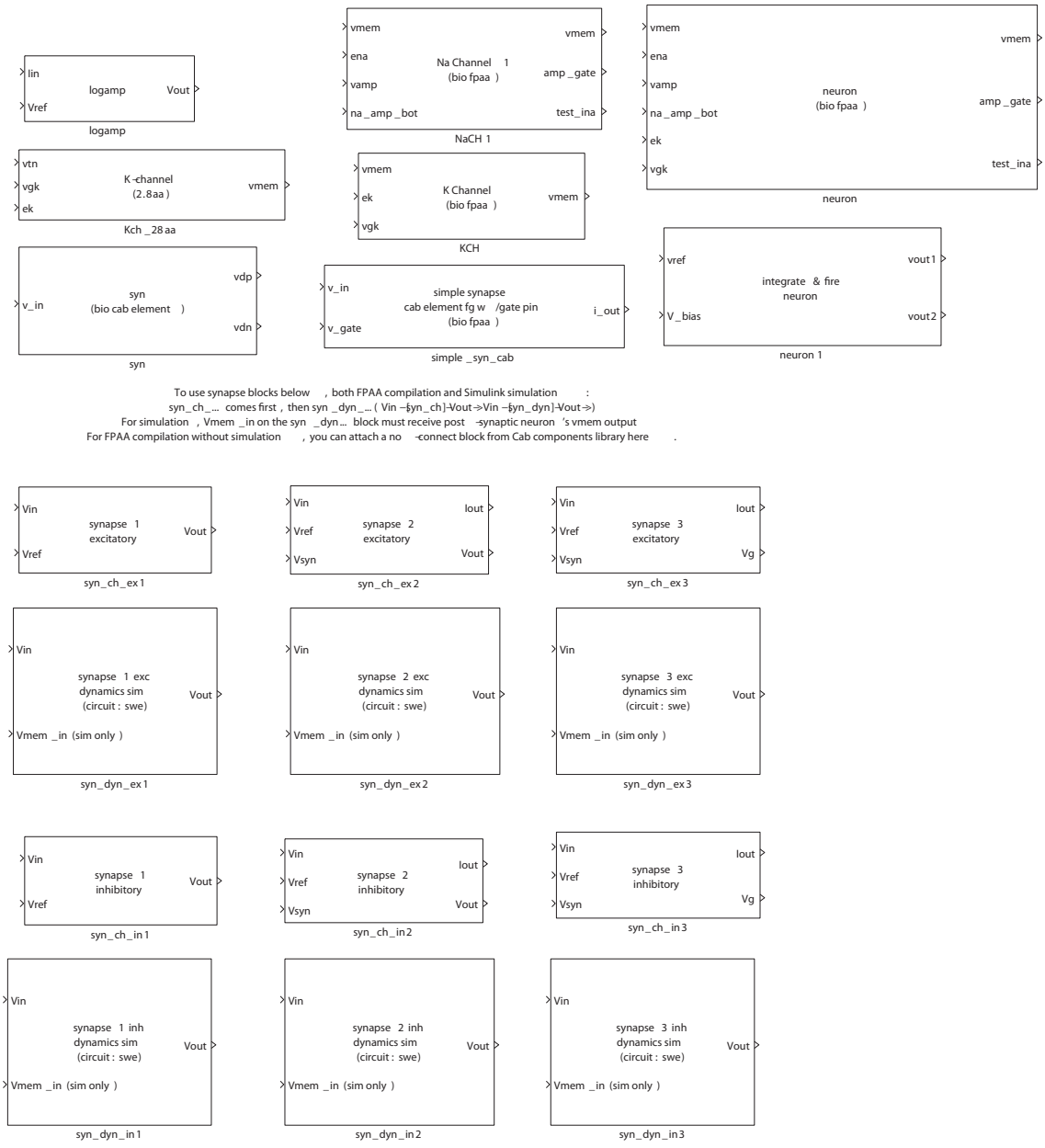


Figure 28. Neurmorphic elements library. Contains neurons and static and adaptive synapses.

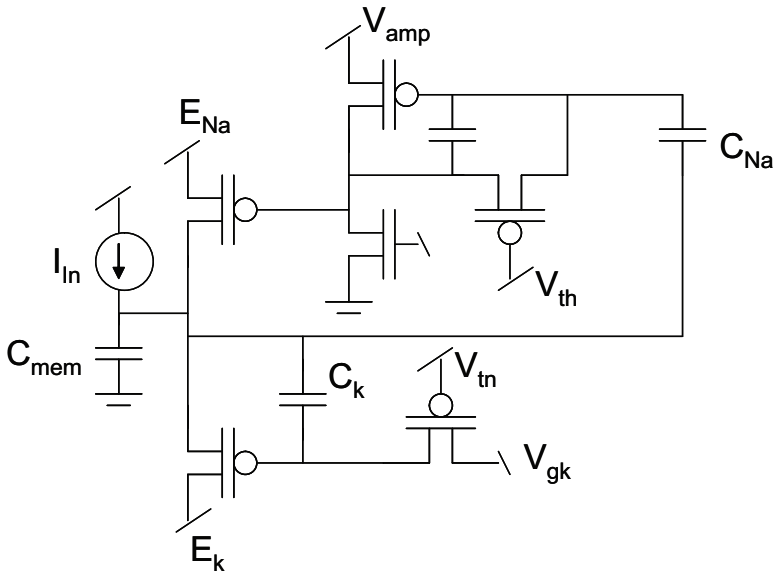


Figure 29. Silicon neuron schematic. This circuit is composed of a Na-channel and K-channel amplifier, with the combined dynamics topologically equivalent to a Hodgkin-Huxley neuron.

dynamical characteristics of transistors. The six-transistor neuron model's sodium and potassium channels are constructed separately and connected together. The sodium channel is a transistor controlled by a bandpass filter type amplifier, giving it dynamics resembling fast activation and slow inactivation. The potassium channel is a modelled as a transistor combined with a low-pass filter type amplifier, resulting in slow activation dynamics. Thus, the combined dynamics of the two channels result in a physically equivalent dynamical system to a Hodgkin-Huxley type neuron [20]. The complete neuron circuit schematic is shown in Figure 29.

Before implementation on the FPAA platform, the circuit was implemented as a custom analog neuron chip [27]. Figure 30 shows the fabricated neuron in a TSMC 0.35 micron CMOS process. The circuit parameters in the chip are meant to be set by off-chip DACs. The input current is supplied by setting the gate of a PMOS transistor whose drain is connected to the membrane potential, Vmem.

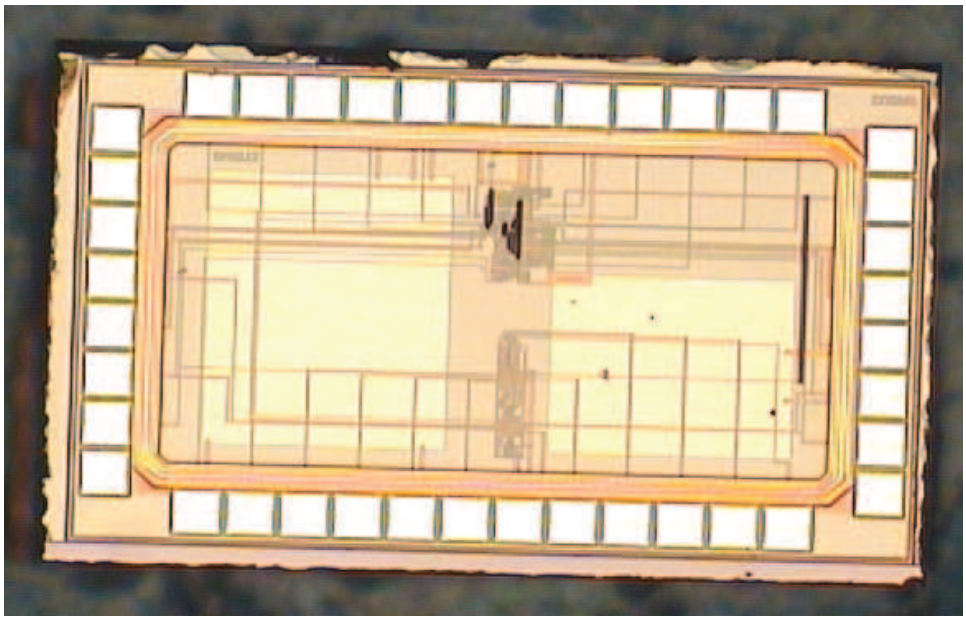


Figure 30. Neuron chip die photo.

5.1.1 Bifurcation Analysis

The dynamics of the neuron model were fully investigated with a bifurcation analysis. The end goal of this type of analysis is an idea of the topological equivalence of the model as compared to other models. In this case, the circuit equations for the six-transistor neuron were derived from the circuit diagram and normalized to give the final dynamical equations for the model as

$$\begin{aligned}
\dot{x} &= e^{-(w+x)}\left\{1-(1+I_{in})\,e^{x-4}\right\}+I_{in}-I_{K0}e^{-y}\\
\dot{y} &= -\frac{I_{\tau n}}{C_K}\left\{e^{y+x}-1\right\}\\
\dot{z} &= \frac{I_{amp}}{8C_Z}\left\{e^{-(z+x)}-1+e^{-(w+x+\beta)}\right\}\\
\dot{w} &= \frac{1.125I_{amp}}{C_Z}\left\{e^{-(z+x)}-1+e^{-(w+x+\beta)}\right\}+\frac{I_{\tau n}}{C_Z}\left\{e^{z+x}-e^{w+x}\right\}
\end{aligned} \tag{4}$$

where I_{K0}=(1+I_{in})(1-e^{-4}).

The expected bifurcation from a Hodgkin-Huxley neuron is a Hopf bifurcation. This type of bifurcation occurs when an equillibrium point for a dynamical system loses stability and becomes a limit cycle [28]. In the Hodgkin-Huxley system, there are actually two subcritical Hopf bifurcation points expected when varying the input current parameter. The

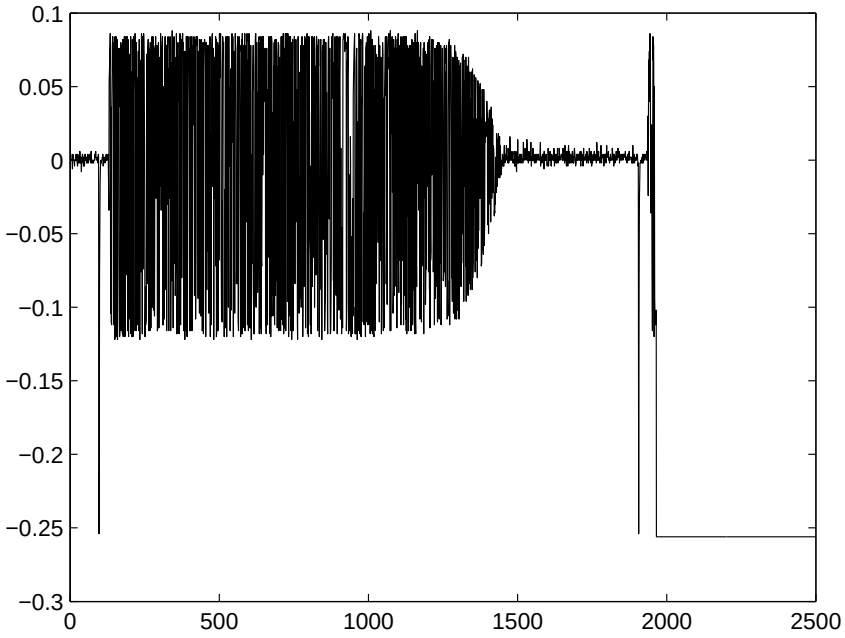


Figure 31. Neuron chip bifurcation diagram showing spiking range as a function of input current. There is a clear transition to and from spiking activity when the neuron undergoes a subcritical Hopf bifurcation.

limit cycle appears at a threshold input current and collapses back to an equillibrium point at a hight threshold input current.

Mathematical analysis of the governing equations for the six-transistor neuron model reveals that it undergoes two subcritical Hopf bifurcation, showing that it is topologically equivalent to a Hodgkin-Huxley neuron model [27]. The fabricated neuron chip was used to test the bifurcation dynamics of the neuron in actual analog hardware. The biasing was done with off-chip DACs, and the input current was varied through the entire range of the expected circuit dynamics. Both bifurcation points were observed in the data, as shown in Figure 31. As the current (x-axis) is increased, the neuron shows a range in which spiking occurs before the membrane voltage becomes constant once again.

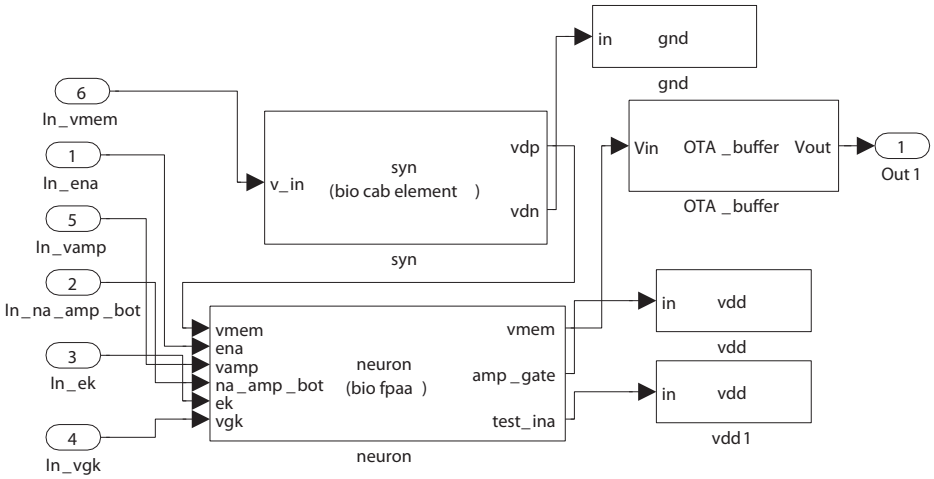


Figure 32. Neuron: Simulink model. The complete example is a spiking Hodgkin-Huxley neuron, input block, and output buffer.

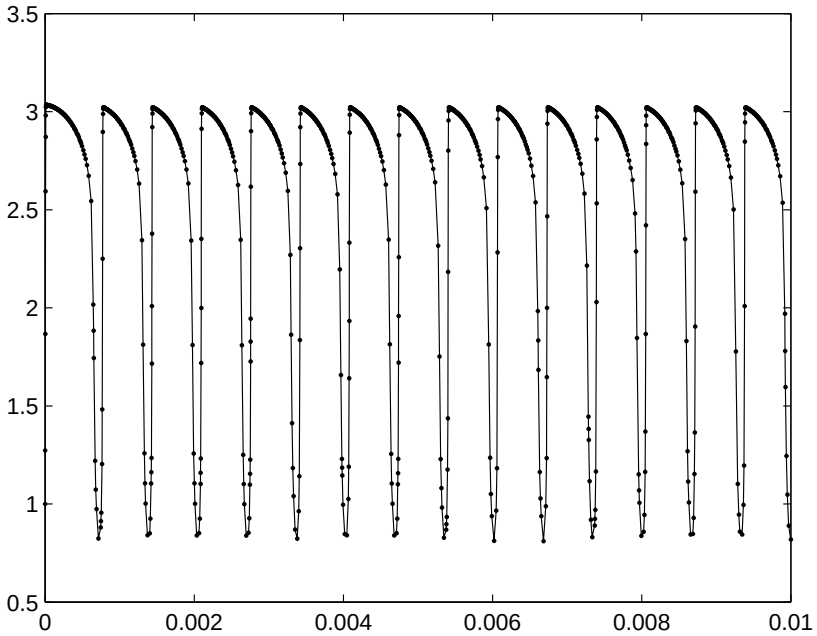


Figure 33. Neuron: Simulink simulation of spiking.

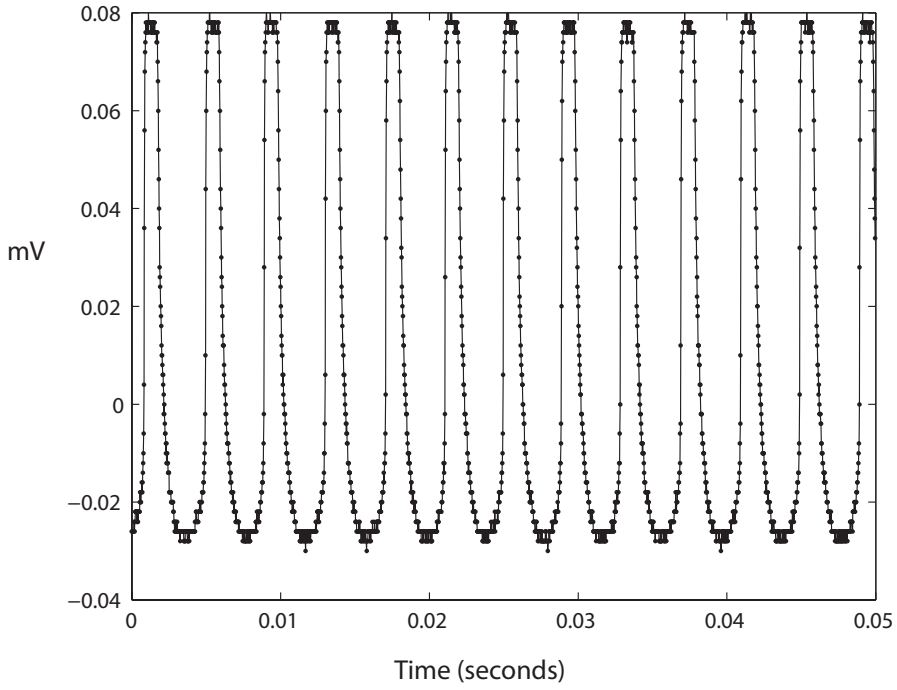


Figure 34. Neuron on FPAA: Spiking activity on chip.

5.2 Neuron on FPAA

The six-transistor Hodgkin-Huxley equivalent neuron was compiled on the RASP 2.9f FPAA using the Sim2spice tool. A Sim2spice library block was constructed for the neuron model and used in a design that includes an input and output port and a buffer for the output, as shown in Figure 32. The block for the neuron also has the correct differential equations for simulating the neuron in Simulink. A simulation of the spiking behavior of the neuron in real-time in Simulink is shown in Figure 33. Finally, the neuron model was compiled to a SPICE circuit netlist and further compiled to the FPAA. It produced spiking behavior as expected, as shown in Figure 34.

5.3 Synchronized Spiking

Two neurons may exhibit synchronized spiking if the input current of one is coupled to the output membrane voltage of the other through a synapse [21]. Figure 35 shows a Simulink

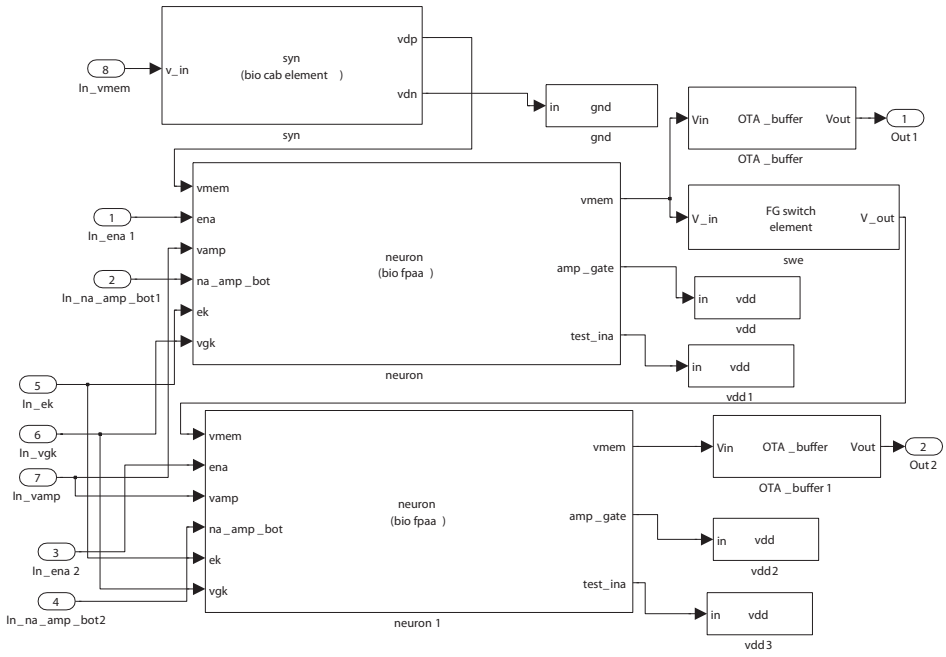


Figure 35. Simulink model of two neurons and a synapse model.

model for two neurons coupled with a simple first-order static synapse model. The system was compiled using Sim2spice and RASPER onto the RASP 2.9f FPAA. Figure 36 shows the spiking activity of the two coupled neurons. They exhibit synchronization in their spiking as expected.

5.4 Adaptive Synapses

Adaptive synapses are a highly desirable feature of neuromorphic engineering. Various learning rules for networks of neurons connected by synapses involve local synaptic modification based on local neural activity [29]. The RASP 2.9f FPAA which was developed specifically with the goal of making configurable adaptive synapses and small adaptive networks of neurons with floating gate elements.

The adaptive CAB circuit elements in the RASP 2.9f, shown in Figure 11, were added to the Sim2spice library and separate blocks were created for each circuit element. The adaptive synapse library is shown in Figure 37. The adaptive synapse block with two

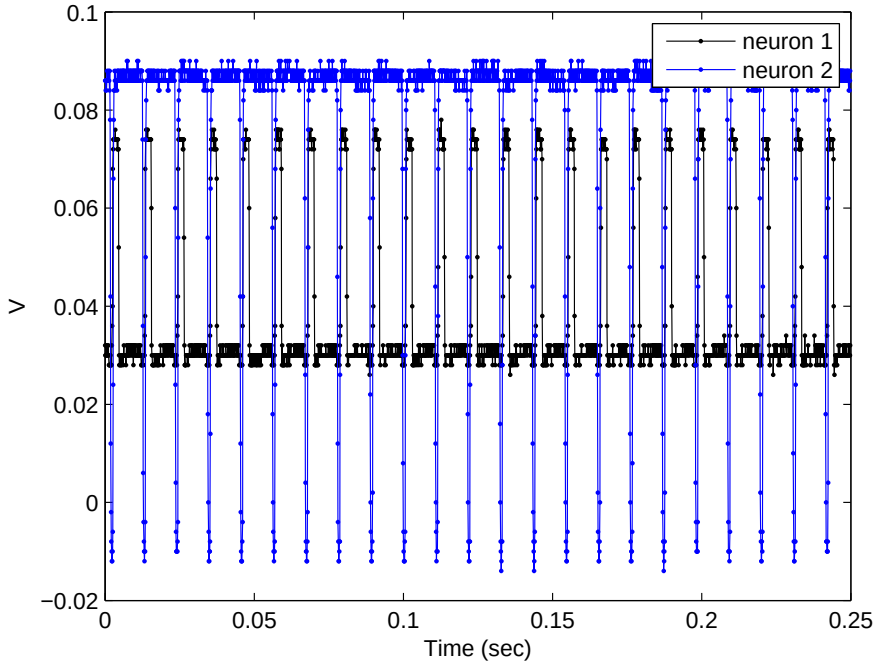


Figure 36. Spiking activity of the two neurons on FPAA. The pair of neurons demonstrates synchronized spiking due to the coupling through the synapse circuit element.

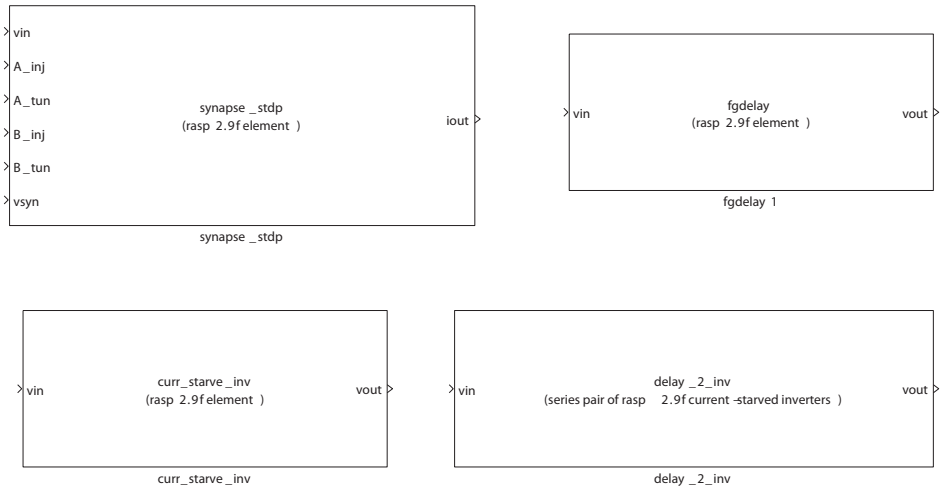


Figure 37. Adaptive synapse library.

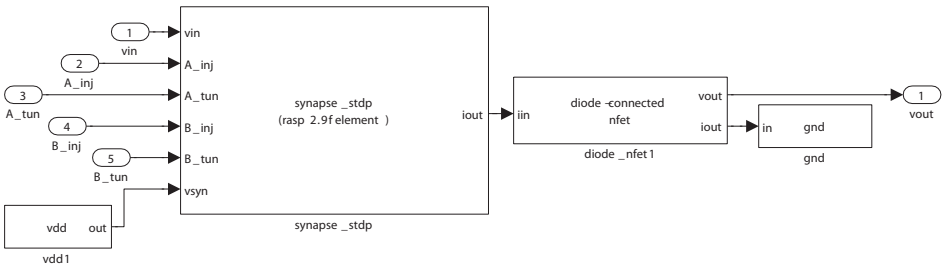


Figure 38. Adaptive synapse test circuit.

inputs was chosen for testing, and Figure 38 shows the full test circuit in Simulink. The output transistor of the synapse block, which is the actual synapse itself, is connected to a diode-connected nfet to transform current to voltage, and this voltage is buffered out.

Once the circuit was compiled on the FPAA, the four digital inputs were connected to DACs on the FPAA board. Both Binj and Btun control signals were set to Vdd (3 volts) and Ainj and Atun were initially set to ground. To test injection, Vdd pulses were applied to the Ainj control line, resulting in millisecond-scale time intervals in which injection was to occur. Figure 39 shows the voltage on the drain of the synapse pFET as a function of time. Each point is a set of ten 30-millisecond injection pulses applied consecutively. The voltage on the pFET increased, showing that charge on the floating gate had correspondingly increased as a result of injection. Figure 40 shows the results of the same experiment but with 100-millisecond injection pulses. The voltage increases in greater steps than in the previous case, corresponding to greater changes in the floating gate charge as a result of longer injection times.

The synapse also contains circuitry for controlling tunneling. Tunneling was tested in a similar manner to injection. The Btun line was pulsed high and the voltage on the drain of the floating gate transistor was measured after each pulse. In the tunneling experiment, each pulse was only applied once. Figure 41 shows the voltage on the drain over time with each point being a 25-millisecond tunneling pulse. The voltage decreased as expected. Figure 42 shows the same experiment with 200-millisecond tunneling pulses. The change

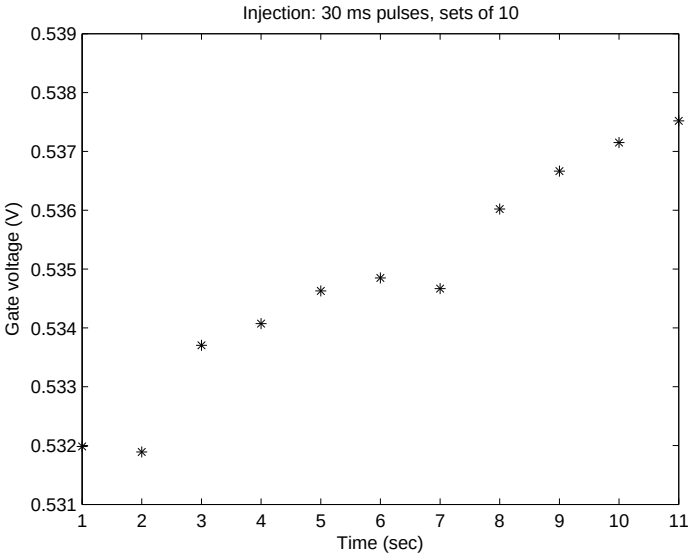


Figure 39. Adaptive injection: 30ms pulses.

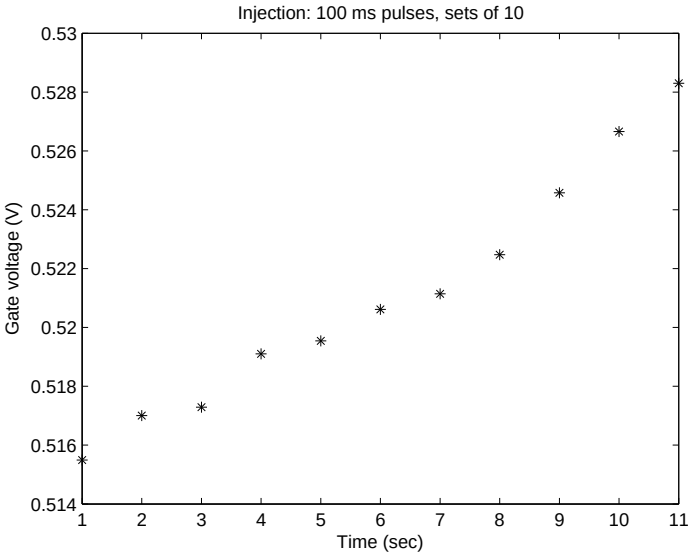


Figure 40. Adaptive injection: 100ms pulses.

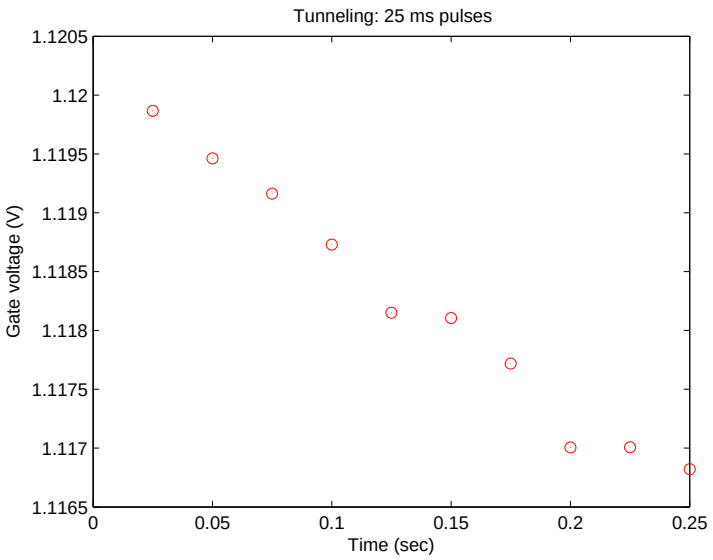


Figure 41. Adaptive tunneling: 25ms pulses.

in floating gate charge is greater for each tunneling pulse.

One very simple synaptic learning rule is Hebbian Learning [30]. The basic idea of this rule is that neurons that fire together, wire together. That is, synaptic strength should grow between those neurons whose action potentials coincide in time. The Simulink model shown in Figure 43 is an attempt to apply the basic idea of this principal in the RASP 2.9f FPAA. The model has two neurons and one adaptive synapse, coupled together in such a way that injection occurs if and only if both neurons are firing simultaneously. This is accomplished by having the output membrane potentials of the two neurons pass through an OTA, extending the firing acivity from approximately 100mv amplitude to a full rail-to-rail swing. Then these converted spiking signals serve as the digital controls for the adaptive synapse. The end result should be spiking that is not synchronized at first and becomes synchronized as the weight grows when the firings overlap.

The Simulink model was successfully compiled to a netlist and FPAA switch list, showing the robustness of the Sim2spice system even when faced with a fairly complex model with many components. The circuit exhibited synchronized activity from the very beginning of the experiment, as shown in Figure 44, making it unclear whether the adaptive

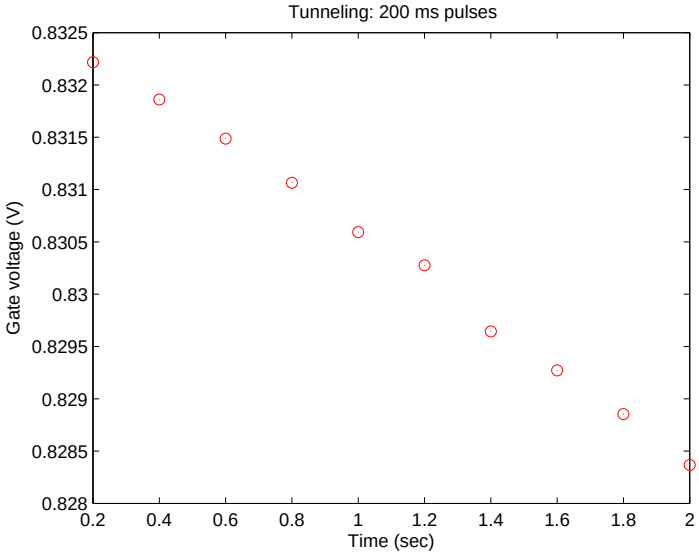


Figure 42. Adaptive tunneling: 200ms pulses.

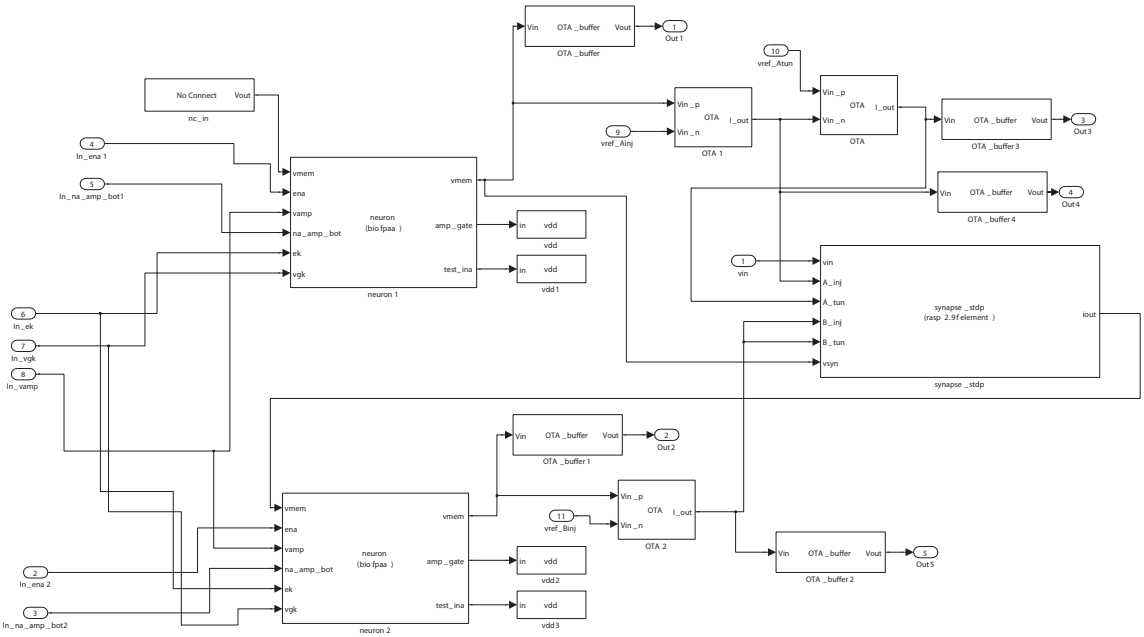


Figure 43. System of two neurons coupled with an adaptive synapse.

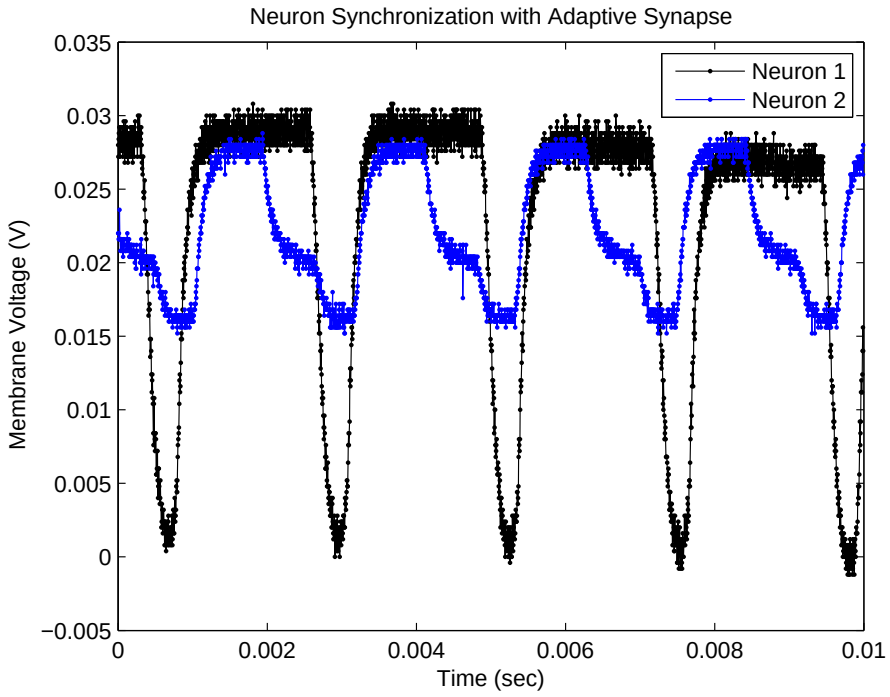


Figure 44. Coupled neurons firing synchronously. It was not established if the adaptive synapse played a role in creating synchrony.

component actually contributed to the synchronous activity or whether the small amount of coupling in a tunneled floating gate transistor was enough to synchronize the two neurons' firings. Further work must be done with this circuit, especially in confirming that the spikes are being transformed to rail-to-rail signals and that adaptation of the synapse is occuring.

5.4.1 Spike-Timing Dependent Plasticity

Spike-timing dependent plasticity (STDP) is a variation on Hebbian learning in which the timing of the pre- and post-synaptic spikes matters in the strength of the change in the synaptic weight. The original work on STDP was done by Bi and Poo, who showed that the change in weight depended approximately exponentially on the time difference between the pre- and post-synaptic spikes, and that if the postsynaptic neuron fired first, the change would be negative, otherwise it would be positive [31]. The basic shape of the STDP curve is shown in Figure 45. This kind of learning rule has been further explored since, and there

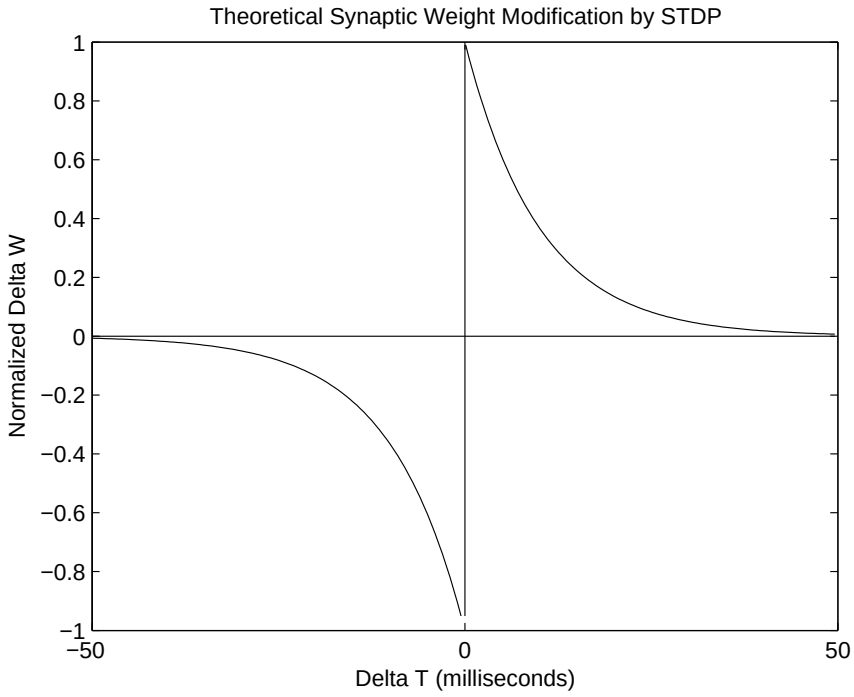


Figure 45. Theoretical Model for Spike Timing Dependent Plasticity (STDP): The change in synaptic weight depends approximately exponentially on the relative timing of the pre- and post-synaptic spikes.

have been some potential practical applications proposed, such as in associative learning for robotic control [32].

One important circuit element that is necessary for setting up the adaptive synapse in the RASP 2.9f FPAA to do STDP is the floating gate delay. This circuit is composed of two current-starved inverters whose current starving transistors are set by floating gates, as shown in Figure 11. A simulink model was implemented, shown in Figure 46, and compiled to the FPAA. Figure 47 shows the output of the delay element in response to a step function. The output is delayed, as desired; however, the circuit had a maximum delay of approximately 3 milliseconds. This delay could potentially be increased by increasing the capacitance on the net connecting the two current-starved inverters, but as this is a single connected CAB element in this FPAA, that work will have to wait until the next revision of this chip.

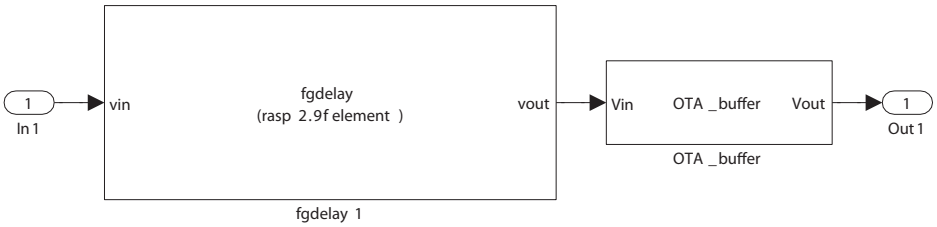


Figure 46. A simulink model for the floating gate delay element.

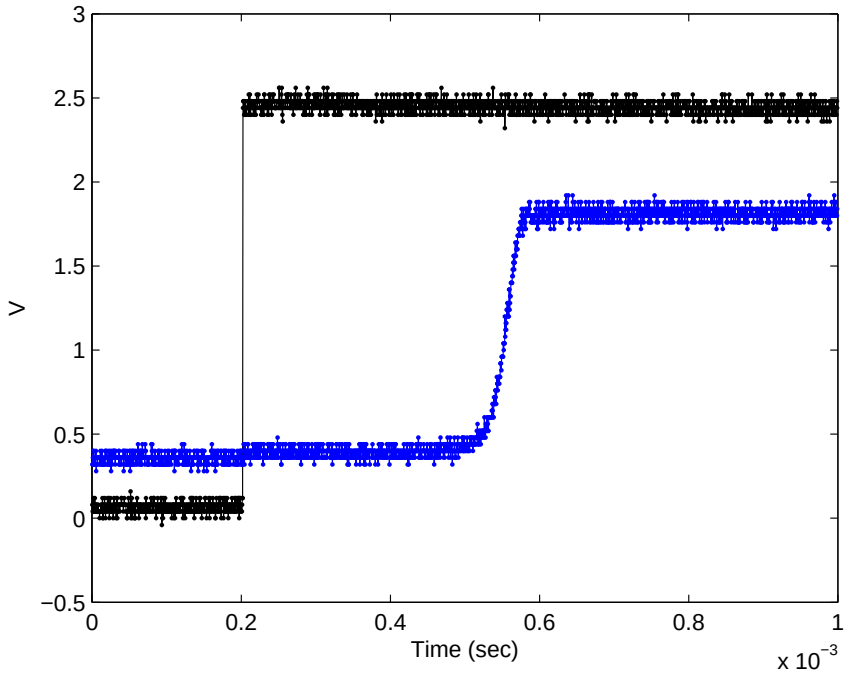


Figure 47. Input step function and delayed output from FPAA for floating gate dedlay element.

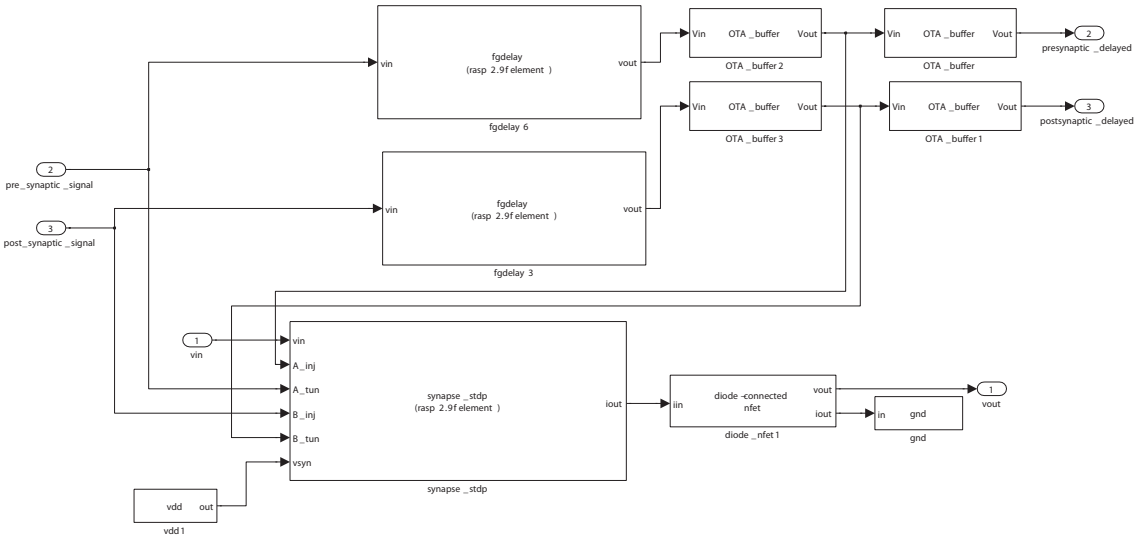


Figure 48. Simulink model for an adaptive synapse testing spikie timing dependent plasticity.

Finally, two delay elements were connected to the adaptive synapse circuit control signals such that the circuit could be tested with timed pulses. If a pulse is applied to the presynaptic input before one is applied to the postsynaptic input, the synapse will undergo injection due to the delay between the pulses. A similar effect will result in tunneling for postsynaptic input pulses before presynaptic input pulses. The Simulink model for this circuit is shown in Figure 48.

This circuit was compiled succesfully to the FPAA and tested. However, possibly due to the delays being very limited, the circuit did not show the appropriate STDP type behavior. Nevertheless, there is now a powerful and intuitive interface for building and testing adaptive neuromorphic circuits on a reconfigurable platform, making it simple to test new variations of learning rules and circuits and diagnose issues. Future versions of this chip will fix the issues that have been encountered and the Sim2spice infrastructure and libraries should save a lot of time and make it simple to test even relatively complex neuromorphic learning rules such as STDP.

CHAPTER 6

CONCLUSION

In this thesis, I have presented a new design tool, Sim2spice, for compiling Matlab Simulink designs to SPICE netlists that can then be further compiled to switch lists for the FPAA.

In Chapter 2, I reviewed the basics of floating-gate transistor operation, the key elements in their design, and how they can be tiled in a two-dimensional array with individual device isolation.

In Chapter 3, I showed some of the FPAA chips that have been developed in the CADSP group and their key design points and possible applications.

In Chapter 4, I explained the tool chain for designing for the FPAA, and how the Simulink to SPICE compilation tool Sim2spice fits into that chain. I showed some library elements that have been created and some example designs to illustrate the use of Sim2spice.

In Chapter 5, I demonstrated the application of Sim2spice to neuromorphic circuits and the adaptive neuromorphic Simulink library that was developed, and attempt to build Simulink and circuit models of adaptive synapse elements.

6.1 Personal Contributions

A lot of the work in this thesis was done by me, but much of it is a result of collaboration with my fellow students in ICElab and CADSP. Most of the initial testing and development of the 2.8a and 2.9a chips was a group effort, involving Arindam Basu, Stephen Brink, Shubha Ramakrishnan, Craig Schlottmann, Scott Koziol and others. The Sim2spice tool was a collaboration between Craig Schlottmann and myself. I developed and maintained the SPICE netlist generator and worked out the interfacing of our blocks with Matlab as S-function blocks, while Craig developed the python script for parsing Simulink model files into Matlab structures. I added the neuromorphic and adaptive component libraries and

wrote the associated part files and Simulink simulation models, with help from Arindam Basu for the correct equations to use for neurons and synapses. Arindam designed and fabricated the neuron chip and RASP 2.9f FPAA. I tested the neuron chip and took the bifurcation data from the hardware, and Arindam derived the equations and theory and did the simulations. I modified the GRASPER device file for the RASP 2.9f FPAA from a file written by others in the group for the 2.9a, and developed all Simulink models for the adaptive synapse and other elements in the chip and took all data from that chip myself.

REFERENCES

[1] G. Franz, "Presentation on gene's law," tech. rep., Georgia Tech, 2008.

[2] D. Kahng and S. Sze, "A floating-gate and its application to memory devices," *The Bell System Technical Journal*, vol. 46, no. 4, pp. 1288–1295, 1967.

[3] C. Twigg, J. Gray, and P. Hasler, "Programmable floating gate fpaa switches are not dead weight," in *Circuits and Systems, 2007. ISCAS 2007. IEEE International Symposium on*, pp. 169 – 172, May 2007.

[4] P. Hasler and J. Dugger, "Correlation learning rule in floating-gate pfet synapse," *Circuits and Systems II: Analog and Digital Signal Processing, IEEE Transactions on*, vol. 48, pp. 65 – 73, 2001.

[5] C. M. Twigg, *Floating Gate Based Large-Scale Field-Programmable Analog Arrays for Analog Signal Processing*. PhD thesis, Georgia Institute of Technology, July 2006.

[6] D. Abramson, *A mite based translinear fpaa and its practical implementation*. PhD thesis, Georgia Institute of Technology, Nov. 2008.

[7] G. Serrano, P. Smith, H. Lo, R. Chawla, T. Hall, C. Twigg, and P. Hasler, "Automatic rapid programming of large arrays of floating-gate elements," in *Circuits and Systems, 2004. ISCAS 2004. IEEE International Symposium on*, 2004.

[8] C. Duffy and P. Hasler, "Modeling hot-electron injection in pfet's," *Computational Electronics, Journal of*, vol. 2, pp. 317 – 322, 2003.

[9] P. D. Smith, M. Kucic, and P. Hasler, "Accurate programming of analog floating-gate arrays," in *IEEE International Symposium on Circuits and Systems*, vol. 5, p. 489492, May 2002.

[10] D. W. Graham, E. Farquhar, B. Degnan, C. Gordon, and P. Hasler, "Indirect programming of floating-gate transistors," *Circuits and Systems I: Regular Papers, IEEE Transactions on*, vol. 54, pp. 951 – 963, May 2007.

[11] T. Hall, C. Twigg, P. Hasler, and D. Anderson, "Developing large-scale field-programmable analog arrays," in *Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International*, p. 142, April 2004.

[12] A. Basu and P. Hasler, "A fully integrated architecture for fast programming of floating gates," in *Circuits and Systems, 2007. ISCAS 2007. IEEE International Symposium on*, pp. 957 – 960, May 2007.

[13] A. Basu, C. Twigg, S. Brink, P. Hasler, C. Petre, S. Ramakrishnan, S. Koziol, and C. Schlottmann, "Rasp 2.8: A new generation of floating-gate based field programmable analog array," in *Custom Integrated Circuits Conference, 2008. CICC 2008. IEEE*, pp. 213 – 216, Sept. 2008.

[14] C. Chang, J. Wawrzynek, and B. Brodersen, "From bee to bee2: Development of supercomputer-in-a-box," in *presentation from Berkeley Wireless Research Center, University of California, Berkeley*, Dec. 2 2004.

[15] B. Sbarcea and D. Nicula, "Automatic conversion of matlab/simulink models to hdl models," in *International Conference on Optimization of Electrical and Electronic Equipment*, 2004.

[16] F. Baskaya, S. Reddy, S. K. Lim, and D. Anderson, "Placement for large-scale floating-gate field-programable analog arrays," in *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 14, pp. 906 – 910, Aug. 2006.

[17] C. Petre, C. Schlottmann, and P. Hasler, "Automated conversion of simulink designs to analog hardware on an fpaa," in *Circuits and Systems, 2008. ISCAS 2008. IEEE International Symposium on*, pp. 500 – 503, May 2008.

[18] P. McGuire, "Pyparsing." http://pyparsing.wikispaces.com/.

[19] C. Petre, C. Schlottmann, and P. Hasler, "Automated conversion of simulink designs to analog hardware on an fpaa." To be published 2009.

[20] A. Basu, C. Petre, and P. Hasler, "Bifurcations in a silicon neuron," in *Circuits and Systems, 2008. ISCAS 2008. IEEE International Symposium on*, 2008.

[21] A. Basu, S. Ramakrishnan, C. Petre, S. Koziol, S. Brink, and P. Hasler, "Neural dynamics in reconfigurable silicon." To be published 2009.

[22] F. Baskaya, B. Gestner, C. Twigg, S. K. Lim, D. Anderson, and P. Hasler, "Rapid prototyping of large-scale analog circuits with field programmable analog array," in *Field-Programmable Custom Computing Machines, 2007. FCCM 2007. 15th Annual IEEE Symposium on*, pp. 319 – 320, April 2007.

[23] S. Koziol, "Field programmable analog array routing and analysis tool users guide." Unpublished, Aug. 2008.

[24] E. Farquhar and P. Hasler, "A bio-physically inspired silicon neuron," *Circuits and Systems I: Regular Papers, IEEE Transactions on*, vol. 52, pp. 477 – 487, 2005.

[25] A. L. Hodgkin and A. F. Huxley, "A quantitative description of membrane current and its application to conduction and excitation in nerve," *Physiology, Journal of*, vol. 117, pp. 500 – 544, 1952.

[26] M. Mahowald and R. Doublas, "A silicon neuron," *Nature*, vol. 354, pp. 515 – 518, 1991.

[27] A. Basu, C. Petre, and P. Hasler, “Dynamics and bifurcations in a silicon neuron.” To be published 2009.

[28] E. M. Izhikevich, *Dynamical Systems in Neuroscience: The Geometry of Excitability and Bursting*. MIT Press, 2006.

[29] L. F. Abbott and P. Dayan, *Theoretical neuroscience: computational and mathematical modeling of neural systems*. MIT Press, 2001.

[30] D. O. Hebb, *The organization of behavior*. Wiley, 1949.

[31] G. Bi and M. Poo, “Synaptic modifications in cultured hippocampal neurons: Dependence on spike timing, synaptic strength, and postsynaptic cell type,” *Neuroscience, Journal of*, vol. 18, pp. 10464 – 10472, 1998.

[32] P. Arena, L. Fortuna, M. Frasca, and L. Patane, “Learning anticipation via spiking networks: Application to navigation control,” *Neural Networks, IEEE Transactions on*, vol. 20, pp. 202 – 216, 2009.